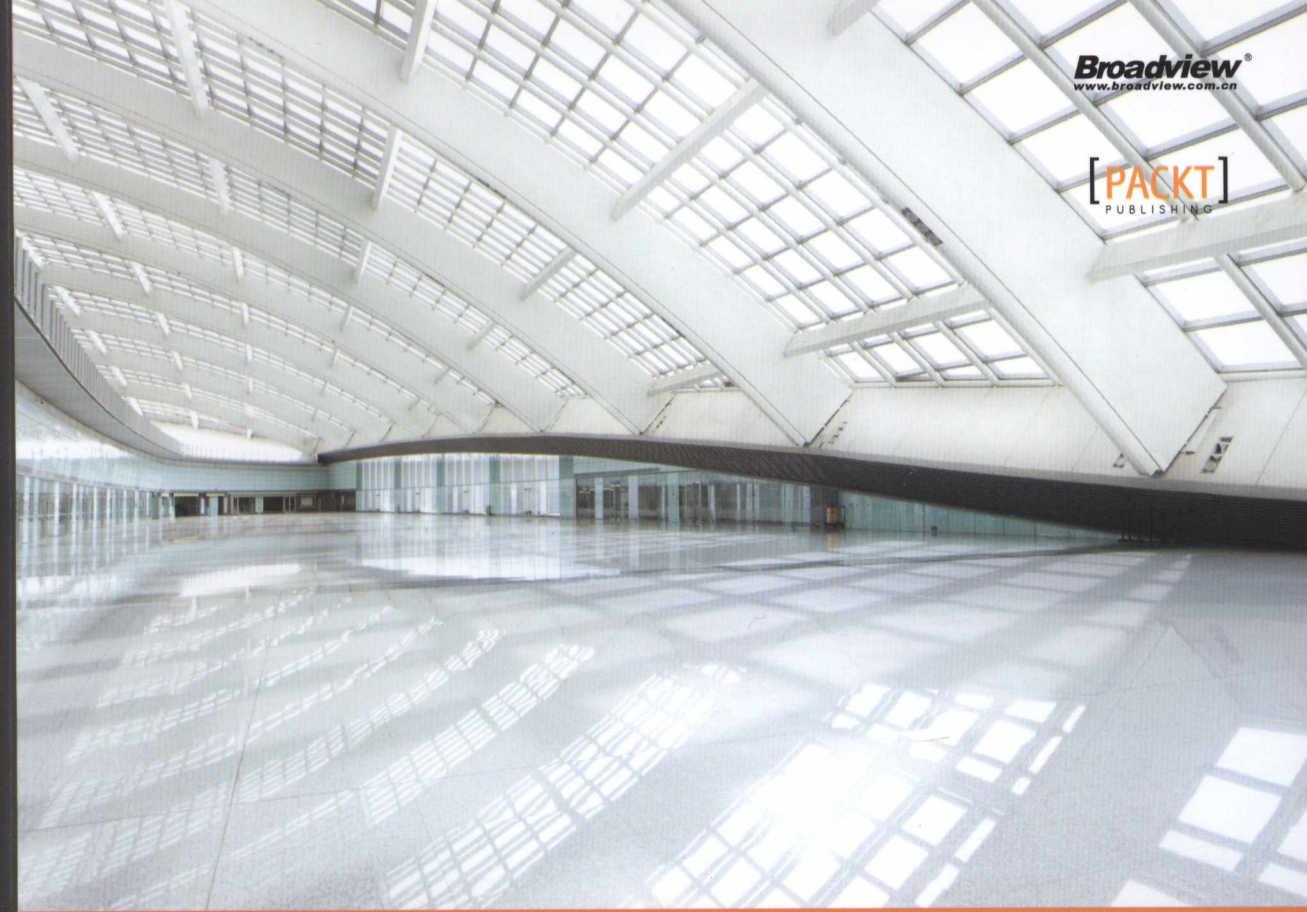
版权注意事项：1、书籍版权归著者和出版社所有；
2、本PDF仅用于个人获取知识，进行私底下知识交流；
3、PDF获得者不得在互联网以任何目的进行传播；
如有需要，请尽量购买正版实体书！支持书籍作者！！



Learning ELK Stack 中文版

使用Elasticsearch、Logstash和Kibana分析日志数据并构建迷人的可视化

[美] Saurabh Chhajed 著
宁海元 张新铭 阚海明 林杰 译

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Learning ELK Stack

Learning ELK Stack 中文版

[美] Saurabh Chhajed 著
宁海元 张新铭 阚海明 林杰 译

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

ELK 技术栈是一套新兴的日志处理开源系统，ELK 分别代表 Elasticsearch、Logstash 和 Kibana。Elasticsearch 是基于 Lucene 构建的一套分布式搜索引擎，提供了实时搜索和聚合分析的能力，是整个 ELK 技术栈的核心。Logstash 构建了数据采集、解析和输出的框架，通过插件可以支持不同的输入、过滤和输出的场景。Kibaba 实现了对 Elasticsearch 的搜索查询和数据可视化。三者组合起来是一套经过很多生产环境验证的日志解决方案。本书针对 ELK 技术栈由浅入深地对每个组件做了介绍，并且通过实际的部署和应用案例，真正地讲清楚了如何利用 ELK 技术栈来实现日志数据的完整管道。

Copyright © 2015 Packt Publishing. First published in the English language under the title'Learning ELK Stack'.

本书简体中文版专有出版权由 Packt Publishing 授予电子工业出版社。未经许可，不得以任何方式复制或抄袭本书的任何部分。专有出版权受法律保护。

版权贸易合同登记号　图字：01-2016-1369

图书在版编目（CIP）数据

Learning ELK Stack 中文版 /（美）苏库拉 • 塞哈特（Saurabh Chhajed）著；宁海元等译. —北京：电子工业出版社，2016.6
书名原文：Learning ELK Stack
ISBN 978-7-121-28884-5

Ⅰ. ①L… Ⅱ. ①苏… ②宁… Ⅲ. ①数据处理－研究 Ⅳ. ①TP274

中国版本图书馆 CIP 数据核字(2016)第 112566 号

责任编辑：张春雨
印　　刷：三河市双峰印刷装订有限公司
装　　订：三河市双峰印刷装订有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×980　1/16　　印张：12　　字数：232 千字
版　　次：2016 年 6 月第 1 版
印　　次：2016 年 6 月第 1 次印刷
定　　价：65.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819　faq@phei.com.cn。

译者序

随着云计算和大数据的发展，分布式架构已经成为常态。分布式架构解决了系统随着业务发展的可扩展性问题，也随之引入了一些新的问题。比如日志，在分布式系统中，日志也随之分布到多台服务器上。这时候，要借助日志来排查系统问题，或者分析业务数据等，成本就比传统的单机系统要高出很多了。

从大数据的角度来看，大数据的来源，主要包括：

1. 数据库
2. 日志文件
3. 爬虫

其中日志文件是最常见也是量最大的数据来源。爬虫也经常会将初步处理的数据以文件的形式存放，也可以归结到日志文件一类。解决日志文件的采集、解析和分析，也是大数据时代的普遍需求。

所以，在云计算和大数据时代，将分布在多台服务器上的日志集中起来，统一地进行存储、解析、搜索和分析展现，成为一种普遍需求。在开源领域，ELK 技术栈是解决日志问题的一种流行方案，这两年来得到了越来越多公司的青睐。ELK 技术栈中的 E，代表 Elasticsearch，是基于 Lucene 构建的一套分布式搜索引擎，并且在传统的基于倒排索引的搜索功能之外，通过引入列式存储 DocValue，具备了不错的分析能力。

我在创立袋鼠云之初，就将日志做为公司大数据产品的主攻方向，基于 Elasticsearch 和 Logstash 开发了袋鼠云日志。在产品的开发过程中，对 Elasticsearch、Logstash 和 Kibana 这套 ELK 技术栈有了更深的认识。在日志场景下，这确实是非常匹配的一套开源的日志解决方案。但要用好这套开源产品，也需要深入理解每个组件的细节，需要有一支技术能力较强的团队来维护和开发。

今年春节前，电子工业出版社的张春雨在朋友圈征集这本书的译者时，刚好是我们团队在设计袋鼠云日志的阶段，所以毫不犹豫地点赞抢得了这个机会。完稿之后，才发现这本书写得真是非常扎实，各种细节娓娓道来，真正做到了深入浅出，对于技术写作来说，这相当不容易。所以这次翻译工作也让我们团队获益不小，希望本书的读者同样也能有所收获。

为了让本书尽快和读者见面，我们组织了一个袋鼠云的专家团队。我本人负责第一、二章，张新铭负责第三、四章，阚海明负责第五、六、七章，林杰负责第八、九、十章，最后由我统一审稿。由于时间比较紧张，疏漏之处难免，敬请读者朋友们指正。

宁海元 袋鼠云 CTO
2016/5/6 于西园

关于作者

Saurabh Chhajed

是一名技术专家，在为产品和服务行业构建企业应用领域，有着丰富而专业的经验。他在大数据分析和机器学习领域有很多实际的构建工程应用的经历，并且乐于布道大数据和 NoSQL 技术。Saurabh 利用他丰富的技术经验，帮助美国很多大型的金融和工业企业从零开始构建大型的产品体系和分布式应用。他在个人网站 `http://saurzcode.in` 分享了很多的个人技术经验。

Saurabh 过去还帮助审阅了 Packt 出版社出版的技术书籍，包括 *Apache Camel Essentials* 和 *Java EE 7 Development with NetBeans 8*。

我想谢谢我的家人——Krati，她对于我非常的支持和鼓励，尽管我为写此书花费了很多本该陪伴她的时间。

我也要感谢本书的技术审稿人和内容编辑，没有他们，本书不可能面世。

关于审稿人

Isra Ellsa 于 2014 年 1 月获得约旦大学计算机本科学士学位。毕业后，她在加州 Santa Clara 的 Seclytics Security 公司做了一年的软件工程师，在这里接触到了多种技术工作。Isra 目前在约旦首都安曼一家名叫 iHorizons 的公司工作。

Anthony Lapenna 的职业生涯是从软件开发开始的，后来转到运维相关的工作，目前是 WorkIT 的系统工程师。他是运维自动化和 DEVOPS 文化的忠实粉丝。他非常热心于追踪最新的技术，并通过撰写技术文章和分享自己的软件活跃于开源生态圈。

Blake Praharaj 是一位软件工程师，精于在忙碌的创业环境驰骋。他目前就职于 Core Informatics 公司，依靠实验室测试和有效的数据解释为多个行业的科学家们提供数据管理解决方案。和任何优秀的开发者一样，他也在不断地学习和探索新的技术！

我想在这里感谢我的另一半，她非常支持和理解我花时间来审阅此书。

我也要感谢 Core Informatics 的整个团队的支持，尤其是 Vico，使得我有时间来学习 ELK 这一技术。

目录

前言 XIII

1 EKL 技术栈介绍 1

日志分析的必要性 1

问题调试 2

性能分析 2

安全分析 2

预测分析 2

物联网日志 3

日志分析的挑战 3

不一致的日志格式 3

不同的时间格式 4

专业知识的需求 5

ELK 技术栈 5

Elasticsearch 5

Logstash 6

Kibana 6

ELK 数据管道 7

安装 ELK 技术栈 8

安装 Elasticsearch 8

运行 Elasticsearch 9

配置 Elasticsearch 10

Elasticsearch 插件 11

安装 Logstash 11

运行 Logstash 12

Logstash 的文件输入插件 13

Logstash 的 Elasticsearch 输出插件 13
配置 Logstash 14
安装 Logstash forworder 15
Logstash 插件 15
安装 Kibana 17
配置 Kibana 18
运行 Kibana 18
Kibana 的界面 19
总结 22
2 构建第一条 ELK 数据管道 23
输入的数据集 23
输入数据集的数据格式 23
配置 Logstash 的输入 25
过滤和处理输入数据 26
将数据存储到 Elasticsearch 29
使用 Kibana 可视化 32
运行 Kibana 32
Kibana 可视化组件 34
构建折线图 35
构建柱状图 36
构建度量 37
构建数据表 38
总结 40
3 使用 Logstash 采集、解析和转换数据 41
配置 Logstash 41
Logstash 插件 42
列出 Logstash 的所有插件 42
插件属性的数据类型 43
Logstash 条件语句 44
Logstash 插件的类型 46
总结 72

4 创建自定义 Logstash 插件 73
Logstash 插件管理 73
插件生命周期管理 74
安装插件 74
更新插件 75
卸载插件 75
Logstash 插件的结构 76
需要的依赖 77
类定义 78
配置插件名字 78
配置选项设置 78
插件方法 79
实现一个 Logstash 过滤器插件 81
构建插件 83
总结 86
5 为什么需要 ELK 中的 Elasticsearch 87
为什么是 Elasticsearch 87
Elasticsearch 的基本概念 88
索引 88
文档 88
字段 89
类型 89
映射 89
分片 89
主分片和副本分片 89
集群 90
节点 90
探索 Elasticsearch API 91
列出所有可用索引 92
列出集群中的所有节点 93
检查集群的健康状态 93

创建索引 94
检索文档 95
删除文档 96
Elasticsearch Query DSL 97
Elasticsearch 插件 104
Bigdesk 插件 104
Elastic-Hammer 插件 105
Head 插件 105
总结 106

6 使用 Kibana 理解数据 107

Kibana 4 的功能 107
搜索词高亮显示 107
Elasticsearch 聚合 108
衍生字段 108
动态仪表盘 108
Kibana 界面 109
搜索页面 109
查询和检索数据 111
总结 116

7 Kibana 可视化和仪表盘 117

可视化页面 117
创建可视化 118
可视化的类型 118
度量和桶聚合 119
可视化 124
仪表盘页面 129
创建新的仪表盘 130
保存和加载仪表盘 131
分享仪表盘 131
总结 132

8 构建完整的 ELK 技术栈 133
输入数据集 133
配置 Logstash 输入 134
访问日志的 Grok 表达式 134
Kibana 可视化 137
运行 Kibana 137
在搜索页进行搜索 139
可视化—图表 141
创建折线图 142
创建区域图 143
创建柱状图 144
创建 Markdown 145
仪表盘页面 146
总结 147
9 生产环境的 ELK 技术栈 149
防止数据丢失 149
数据保护 150
系统可扩展性 152
数据保留 153
ELK 技术栈实施案例 153
LinkedIn 的 ELK 技术栈 153
SCA 使用 ELK 的案例 156
SCA 如何使用 ELK 157
如何帮助分析 157
SCA 使用 ELK 做监控 158
Cliffhanger Solutions 使用 ELK 的案例 158
Kibana 示例——Packetbeat 仪表盘 160
总结 163
10 扩展 ELK 165
Elasticsearch 插件和工具 165

用于索引管理的 Curator 165
用于安全的 Shield 167
用于监控的 Marvel 169
ELK 的路线图 172
Elasticsearch 路线图 172
Logstash 路线图 172
Kibana 路线图 173
总结 174

前言

本书旨在介绍如何使用开源的 Elasticsearch、Logstash 和 Kibana 技术栈来构建企业自己的 ELK 数据管道。本书也覆盖了每个技术组件的核心概念，以及怎样快速使用这些技术搭建日志分析解决方案。本书一共分为十章，其中第 1 章主要介绍如何快速地部署 ELK 技术栈的各个组件，这样在第 2 章中就可以开始构建第一个数据管道；第 3 章到第 7 章分别详细介绍了技术栈各个组件的详情；第 8 章则使用 ELK 构建了一个完整的数据管道；第 9 章则介绍了 ELK 技术栈的几个实际使用案例；最后，在第 10 章中也涉及了如何利用一些周边工具来提升 ELK 技术栈的能力。

本书的内容

第 1 章，主要说明了 ELK 技术栈的概念，以及它能够解决的问题。本章介绍了 ELK 技术栈中每个组件的功能，也涵盖了如何安装 Elasticsearch、Logstash 和 Kibana 组件的内容，可以帮助用户快速运行起整个技术栈。

第 2 章，在这一章构建了一个基本的 ELK 数据管道，输入的数据源是 CSV 格式的文件。通过这个例子中对 ELK 基本配置项的解读，可以帮助我们完成 ELK 搭建并快速地开始分析日志。

第 3 章，覆盖了 Logstash 的关键特性，也说明了如何集成各种不同的输入和输出源。本章也比较细致地介绍了 Logstash 的输入、过滤和输出插件，来帮助完成数据的采集、解析、转换和传输。

第 4 章，解释了在标准插件无法满足时，如何为多样化的需求创建自定义的 Logstash 插件。本章描述了 Logstash 插件的生命周期，以及如何开发和发布不同类型的输入插件、过滤插件和输出插件。

第 5 章，描述了 ELK 技术栈中的 Elasticsearch 的作用、基本概念和关键特性，比如索

引、文档、分片、集群，等等。另外也包含了 Elasticsear 中不同的索引和搜索 API，以及查询 DSL。

第 6 章，描述了如何利用 Kibana 来搜索、查看存储在 Elasticsearch 索引中的数据，以及如何对着写数据进行实时的交互操作。本章探索了各种不同的搜索选项，以及如何使用 Kibaba 界面的搜索页功能。

第 7 章，通过一些案例来描述如何使用 Kibana 可视化和仪表盘的细节。同时也对设置页做了一些说明，包括配置索引模式、衍生字段，等等。

第 8 章，演示了如何整合 ELK 技术栈的三个技术组件，来构建完整的数据管道，也再次解释了前面章节中提过的各组件的功能。

第 9 章，解释了在生产环境中使用 ELK 技术栈需要注意的关键点。并且也提供了一些不同生产环境中的 ELK 技术栈实际案例。

第 10 章，描述了通过联合使用 ELK 的周边工具，来扩展技术栈的能力。

阅读本书需要准备什么

- Unix 操作系统（任意喜欢的版本都可以）
- Elasticsearch 1.5.2
- Logstash 1.5.0
- Kibana 4.0.0032

本书面向的读者

本书适合任何希望低成本分析数据的人。读者不一定需要有 ELK 技术栈或者某些组件的背景知识，但熟悉 NoSQL 数据库或者掌握一些编程基础知识会有帮助。

约定

本书中使用了很多格式的文本，以区分各种不同的信息。这里我们举例说明这些格式，并解释它们的含义。

在文本、数据库表名、文件夹名称、文件名、文件扩展名、路径名、伪 URL、用户输入和 Twitter 中，格式是这样的，例如："上面的命令会安装 `rabbitmq` 输入插件。"

代码格式如下：

```
filter {
drop {
}
}
```

编写任何命令行输入或输出如下：

```
$bin/plugin install logstash-input-rabbitmq
```

新术语和**重要词汇**以黑体显示。在屏幕上看到的词，例如，在菜单或对话框上，你这样表示："现在点击 **Next** 按钮，将移动到下一屏幕。"



警告或重要提示以这种方式表示。



技巧和窍门像这样表示。

下载示例代码

你可以从 `http://www.broadview.com.cn` 下载所有已购买的博文视点书籍的示例代码文件。

勘误表

虽然我们已经尽力谨慎地确保内容的准确性，但错误仍然存在。如果你发现了书中的错误，包括正文和代码中的错误，请告诉我们，我们会非常感激。这样，你不仅帮助了其他读者，也帮助我们改进后续的出版。如发现任何勘误，可以在博文视点网站相应图书的页面提交勘误信息。一旦你找到的错误被证实，你提交的信息就会被接受，我们的网站也会发布这些勘误信息。你可以随时浏览图书页面，查看已发布的勘误信息。

1

EKL 技术栈介绍

本章主要解释了在今天这个数据驱动的世界中日志分析的重要性，以及伴随日志分析而来的挑战。本章介绍了如何使用 ELK 技术栈作为完整的日志分析解决方案，并解释了什么是 ELK 技术栈，以及每个开源组件包括 Elasticsearch、Logstash 和 Kibana 所扮演的角色。此外，本章还简要地介绍了每个组件的关键特性，并描述了安装和配置的步骤。

日志分析的必要性

日志可以为我们提供关于系统行为的必要信息。然而，每个不同的服务，或者同一个系统中不同的组件中，日志的内容和格式都可能是不同的。比如说，一个扫描器可能会记录同其他设备通信时发生的相关错误；另一方面，Web 服务器则会记录所有进入的请求、返回的响应，以及响应时间等信息。同样的，一个电子商务网站的应用日志也会记录业务特定的日志。

正因为日志的内容是多样化的，它们的用处才能体现出来。例如，上述扫描器的日志可以用来排查故障、执行简单的状态检查或者生成报表，而 Web 服务器日志则可以用来分析跨多个产品之间的流量模式。通过电子商务网站的日志可以分析出某个特定位置发出的包裹是否被频繁地退回，还能分析出可能的原因是什么。

下面是一些可以使用日志分析的常见用例：

- 问题调试
- 性能分析

- 安全分析
- 预测分析
- 物联网（IoT）日志

问题调试

问题调试是在应用程序中开启日志最常见的理由之一。调试日志最简单和最频繁的用法是查找特定的错误消息或者发生的事件。如果一名系统管理员觉得程序是因为网络失败而崩溃的，他可以在服务器日志中尝试查找类似 `connection dropped` 这样的消息来分析具体的原因。一旦 bug 或者问题被定位，日志分析解决方案可以帮助捕获应用的信息，并且提供问题发生时的日志快照给开发团队用于后续的深入分析。

性能分析

日志分析有助于优化或者调试系统的性能，可以提供系统瓶颈处的关键信息。了解系统的性能往往是在了解系统中如何使用资源。日志可以帮助分析系统中单个资源的使用情况、应用程序中的多线程行为，以及潜在的死锁条件等。日志中一般会带有时间戳，这是分析系统随时间推移的行为变化所必不可少的信息。例如，可以通过 Web 服务器日志中的响应时间和 HTTP 响应代码来了解每个服务的情况等。

安全分析

对任何组织来说，日志都可以在应用安全的管理中起到关键的作用，特别有助于检测安全漏洞、应用程序滥用、恶意攻击等。当用户与系统进行交互时会产生日志事件，可以帮助跟踪用户行为、识别可疑的活动，并提供警报或记录安全事件的违规行为。

入侵检测过程也包括日志本身的会话重构。例如，系统中的 `SSH` 登录事件可以用来识别机器上的各种侵入行为。

预测分析

预测分析是近年来的研究热点之一。日志和事件数据可用于非常准确的预测分析。预测分析模型有助于识别潜在客户、规划资源、管理和优化库存、提升工作负载和资源调度的效率。它也有助于指导营销策略、用户人群定向、广告投放策略等。

物联网日志

当涉及到物联网设备（设备或机器在没有人工干预的情况下进行交互）时，对系统进行监测和管理，以保持停机时间最短，并快速解决任何重大 bug 或问题来说，日志是非常关键的。由于这些设备的位置可能分布在一个非常大的地理范围内，所以需要尽可能少的人工介入。我们期望日志数据对于了解系统的行为、减少停机时间发挥重要的作用。

日志分析的挑战

目前的日志分析过程主要是检查在多个服务器上的日志，这些日志是应用程序中不同的的组件和系统所记录的。这有各种各样的问题，分析这些日志是一个既费时又烦琐的工作。让我们来看一些常见的问题：

- 不一致的日志格式
- 离散的日志
- 专业知识的需求

不一致的日志格式

每一个应用或者设备都有自己记录日志的方式，每种格式都需要各自的专家才能解读。并且在不同格式的日志之间进行搜索也是非常困难的。

接下来让我们来看一些常见的日志格式。注意观察一个有趣的现象，不同的日志可能采用不同的时间戳格式，也有不同的方式来表示信息（INFO）和错误（ERROR）等，并且在日志中记录这些组件的顺序也不尽相同。如果只是看一眼日志记录，很难弄清楚究竟是什么地方出现了什么问题。这就是像 Logstash 这样的工具可以提供帮助的地方。

Tomcat 日志

典型的 Tomcat 服务器启动的日志记录如下：

```
May 24, 2015 3:56:26 PM org.apache.catalina.startup.HostConfig deployWAR
INFO: Deployment of web application archive \soft\apache-tomcat-7.0.62\
webapps\sample.war has finished in 253 ms
```

Apache 访问日志-组合日志格式

典型的 Apache 访问日志格式如下：

```
127.0.0.1 - - [24/May/2015:15:54:59 +0530] "GET /favicon.ico HTTP/1.1"
200 21630
```

IIS 日志

典型的 IIS 日志如下：

```
2012-05-02 17:42:15 172.24.255.255 - 172.20.255.255 80 GET /images/
favicon.ico - 200 Mozilla/4.0+(compatible;MSIE+5.5;+Windows+2000+Server)
```

不同的时间格式

对于不同的应用和不同的设备产生的日志，不仅仅是日志格式本身不一样，就连其中的时间戳格式也可能不同。系统中不同组件日志中的时间戳格式不同，也使得很难将多个系统在同一时间发生的事件关联起来：

- 142920788
- Oct 12 23:21:45
- [5/May/2015:08:09:10+0000]
- Tue 01-01-2009 6:00
- 2015-05-30 T 05:45 UTC
- Sat Jul 23 02:16:57 2014
- 07:38, 11 December 2012 (UTC)

离散的日志

在应用程序中，日志往往分布在不同的服务器和不同的组件中。多个组件在多个位置记录日志，增加了日志分析的复杂性。如果只有一两台服务器，想从日志中查找信息直接运行 `cat` 或者 `tail` 命令即可，或者也可以通过管道使用 `grep` 命令。但是如果你有 10 台、20 台，或者说 100 台服务器呢？使用上述命令来搜索日志是无法推广到这样庞大的机器集群的，这时候就需要一个集中的日志管理和分析的解决方案。

专业知识的需求

通常来说，人们关心的是业务相关的信息，而不是日志本身。很多人没有必要直接访问日志，也缺乏从日志中以最快的方式得到所需信息的技术能力。这种情况使得分析问题变得很慢，甚至很多时候根本无法分析。

ELK 技术栈

ELK 平台是一个完整的日志分析解决方案，由这三个开源工具构建而成：Elasticsearch、Logstash 和 Kibana。它尝试来应对前面一节我们所提到的问题和挑战。ELK 使用了开源技术栈让 Elasticsearch 用于深度搜索和数据分析；Logstash 用于日志集中管理，包括从多台服务器上传输和转发日志，并对日志进行丰富和解析；最后是 Kibana，提供了强大而美观的数据可视化。ELK 技术栈目前主要由 Elastic（之前叫 Elasticsearch）公司维护和支持。

让我们看一下这三个系统的简单介绍：

- Elasticsearch
- Logstash
- Kibana

Elasticsearch

Elasticsearch 是一个基于 Apache Lucene 的分布式开源搜索引擎，使用 Apache2.0 开源协议发布（这意味着你可以免费下载、使用或者修改）。它在 Lucene 的实时搜索之外提供了可扩展性、可靠性和多租户功能。Elasticsearch 的这些功能可以通过基于 JSON 的 RESTful API 来使用。它的搜索能力来源于后端的 Apache Lucene 引擎，无须预先定义数据结构就能动态地对数据进行索引。Elasticsearch 能够快速响应搜索请求是因为使用了索引来进行文本搜索。

很多大公司都使用了 Elasticsearch，包括 Github、SoundCloud、FourSquare、Netflix，以及很多其他著名的公司。下面例举了一些典型的用户案例：

- **Wikipedia**：使用 Elasticsearch 来提供文本全文搜索，以及一些产品功能，如一边输入一边搜索(search-as-you-type)和搜索建议(did-you-mean)。

- **The Guardian**：使用 Elasticsearch 处理每天的 4000 万个文档，提供跨组织的网站流量实时分析，帮助公司更好地理解用户需求。
- **StumbleUpon**：使用 Elasticsearch 提供跨平台智能搜索，并为数百万客户提供更好的推荐。
- **SoundCloud**：使用 Elasticsearch 为数以百万计的用户提供跨地域的实时搜索。
- **GitHub**：使用 Elasticsearch 对超过 800 万行的代码库和跨平台事件进行索引，以便提供实时搜索的功能。

Elasticsearch 的关键特性包括：

- 它是一个开源的分布式、可扩展和高可用的实时文档存储。
- 提供了实时搜索和分析的能力。
- 提供了复杂的 RESTful API，包括查找和其他各种功能，如批量搜索、地理位置搜索、自动完成、上下文搜索建议和结果片段等。
- 能够很简单地进行水平扩展，也很容易和其他云基础设施如 AWS 等集成。

Logstash

Logstash 是一个数据管道，主要用来收集、解析和分析大量结构化和非结构化的数据以及各种系统产生的事件。Logstash 提供了输入插件来支持不同的数据源和平台，设计用来高效地处理日志、事件和非结构化数据源，然后通过输出插件如文件、标准输出（如输出到运行 Logstash 的控制台）或者 Elasticsearch 等输出结果数据。

Logstash 的关键特性如下：

- **集中化的数据处理**：Logstash 通过数据管道，可以集中化地处理数据。使用不同的输入和输出插件，可以将各种不同的输入源转换成一种单一的通用格式。
- **支持自定义日志格式**：不同应用生成的日志通常有不同的特殊格式。Logstash 可以分析和处理大规模的自定义格式的日志。Logstash 自带了很多开箱即用的过滤插件，也支持用户编写自定义的插件。
- **插件开发**：可以开发和发布自定义插件。实际上现在已经有很多可用的自定义插件。

Kibana

Kibana 是一个基于 Apache2.0 开源协议的开源数据可视化平台。它可以对存储于

Elasticsearch 的索引中的各种结构化和非结构化的数据进行可视化呈现。Kibana 完全使用 HTML 和 Javascript 编写，它利用 Elascticsearch 的 RESTful API 来实现其强大的搜索能力，将结果显示为各种震撼的图形提供给最终的用户。从基本的商业智能到实时调试等不同场景，Kibana 都可以通过直方图、地图、饼图、其他图形和表格等方式来表达数据。

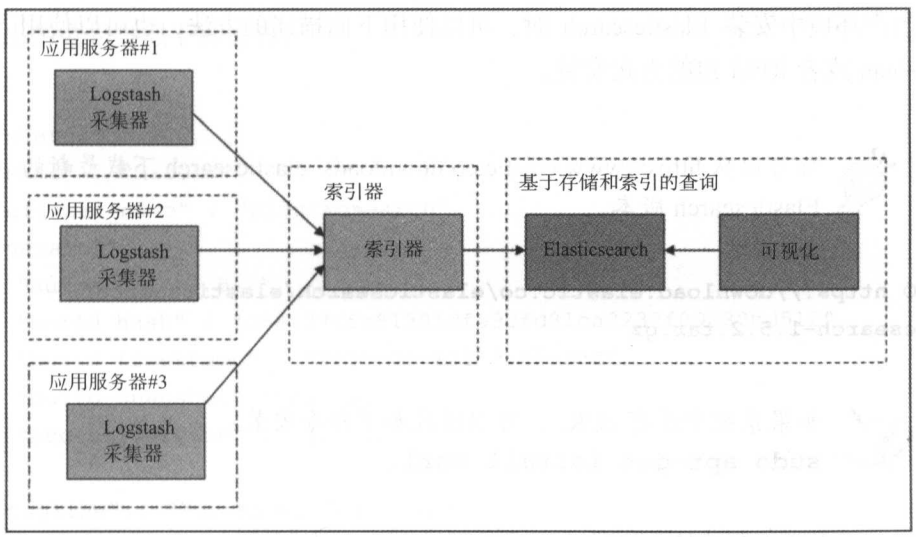
Kibana 使得理解大量的数据变得容易。其基于浏览器的简单界面，使得用户可以很迅速地创建和分享动态的仪表盘，以呈现 Elasticsearch 的实时搜索结果。

Kibana 的关键特性如下：

- 它为商业智能提供了一个灵活的分析和可视化平台。
- 它提供了实时分析、汇总、图表和调试的能力。
- 它提供了一个直观且用户友好的界面，并且是高度可定制的，可以根据需要对图表进行拖放和对齐。
- 可以管理多个仪表盘，并且能够保存。仪表盘可以在多个系统中分享和嵌入。
- 可以分享日志搜索结果的快照，可以隔离不同的问题处理过程。

ELK 数据管道

一个典型的 ELK 技术栈的数据管道看起来如下图所示：



在典型的 ELK 技术栈的数据管道中，多个应用服务器上的日志通过 Logstash 采集器传输到一个集中化的索引器中，索引器将处理后的数据结果输出到 Elasticsearch 集群，然后 Kibana 通过查询 Elasticserch 集群中的日志数据创建仪表盘，做可视化展现。

安装 ELK 技术栈

ELK 技术栈需要 Java 运行环境，建议安装最新的 Java 版本。在写作本书时，ELK 要求的最低版本是 Java 7。你可以使用 Oracle 的官方版本，也可以使用如 OpenJDK 这样的开源版本。

可以在 Shell 中执行如下命令来确认 Java 的版本：

```
> java -version
java version "1.8.0_40"
Java(TM) SE Runtime Environment (build 1.8.0_40-b26)
Java HotSpot(TM) 64-Bit Server VM (build 25.40-b25, mixed mode)
```

如果确认系统中已经安装好 Java，接下来就可以开始安装 ELK。

安装 Elasticsearch

在生产环境中安装 Elasticsearch 时，可以使用下面描述的方法，也可以使用下载页提供的 Debian 或者 RPM 包的方式安装。



你可以从 https://www.elastic.co/downloads/ elasticsearch 下载最新的 Elasticsearch 版本。

```
curl -O https://download.elastic.co/elasticsearch/elasticsearch/
elasticsearch-1.5.2.tar.gz
```



如果系统中没有 cURL，可以通过如下命令安装：
sudo apt-get install curl。

然后，在本地文件系统中解压 zip 文件：

```
tar -zxvf elasticsearch-1.5.2.tar.gz
```

然后，直接进入安装目录：

```
cd elasticsearch-1.5.2
```



Elastic 是 Elasticsearch 背后的商业公司，最近发布了 Elasticsearch2.0 版本，主要特性包括引入一些新的聚合函数、更好的压缩选项、通过合并 query 和 filter 简化 DSL，并且大幅度地提升了性能，更详细的信息请参考官方文档：https://www.elastic.co/ guide/en/elasticsearch/reference/current/index. Html。

运行 Elasticsearch

运行如下命令即可运行 Elasticsearch：

```
$ bin/elasticsearch
```

使用-d 参数可以以后台守护进行的方式运行。

测试的话，只需要在新的终端窗口中执行如下命令：

```
curl 'http://localhost:9200/?pretty'
```

输出的结果类似下面这样：

```
{
  "status" : 200,
  "name" : "Master",
  "cluster_name" : "elasticsearch",
  "version" : {
    "number" : "1.5.2",
    "build_hash" : "c88f77ffc81301dfa9dfd81ca2232f09588bd512",
    "build_timestamp" : "2015-05-13T13:05:36Z",
    "build_snapshot" : false,
    "lucene_version" : "4.10.3"
  },
  "tagline" : "You Know, for Search"
}
```

如果要关闭 Elasticsearch，只要像下面这样调用 API：

```
curl -XPOST 'http://localhost:9200/_shutdown'
```

配置 Elasticsearch

Elasticsearch 的配置文件一般放置在安装目录下的 config 目录中。config 目录下有两个文件，分别是 elasticsearch.yml 和 logging.yml。前者用来配置 Elasticsearch 不同模块的属性，比如网络地址、路径等，后者则用来配置自身的日志记录选项。

配置文件是 YAML 格式的，下面简要地介绍一些配置参数。

网络地址

指定网络相关模块的绑定和发布地址：

```
network :
    host : 127.0.0.1
```

路径

指定数据和日志文件的路径：

```
path:
  logs: /var/log/elasticsearch
  data: /var/data/elasticsearch
```

集群名

指定生产集群的名字，集群将根据这个名字来自动发现和加入节点：

```
cluster:
  name: <NAME OF YOUR CLUSTER>
```

节点名

指定每个节点的默认名字：

```
node:
  name: <NAME OF YOUR NODE>
```

Elasticsearch 插件

Elasticsearch 有各种插件，可以简化诸如管理索引、集群等任务。其中一些常用的插件有 Kopf、Marvel、Sense、Shield 等，在接下来的章节里将会介绍，这里我们先来看一下 Kopf 插件。

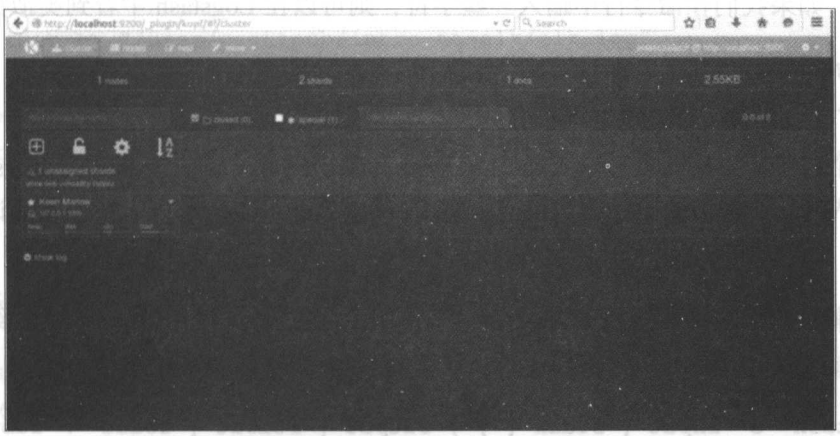
Kopf是用 Javascript、AngularJS、Jquery 和 Twitter Bootstrap 写的一个简单的 Elasticsearch Web 管理工具，简化了管理 Elasticsearch 集群的常见任务。这个插件没有覆盖所有的 API，但提供了一个 REST 客户端，允许你探索 Elasticsearch API 的全部功能。

为了安装 `elasticsearch-kopf` 插件，需要在 Elasticsearch 的安装目录执行如下命令：

```
bin/plugin -install lmenezes/elasticsearch-kopf
```

然后访问如下地址：http://localhost:9200/_plugin/ kopf/。

你会看到一个类似下图的页面，展示了 Elasticsearch 的节点、分片、文档、所占空间大小等信息，并且可以查询文档的索引。



Elasticsearch Kopf 界面

安装 Logstash

首先，从官网下载页面下载最新的 Logstash 安装文件。



如下地址可以查看 Logstash 的最新发行版本
https://www.elastic.co/downloads/logstash。

```
curl -O http://download.elastic.co/logstash/logstash/logstash-1.5.0.tar.
gz
```

然后在本地文件系统中解压缩：

```
tar -zxvf logstash-1.5.0.tar.gz
```

接下来就可以使用最基本的配置来运行 Logstash 了。

运行 Logstash

使用-e 参数指定以标准输入和标准输出的方式来运行 Logstash：

```
cd logstash-1.5.0
bin/logstash -e 'input { stdin { } } output { stdout {} }'
```

现在，如果我们在命令行中输入一些字符，则可以在 Logstash 中看到类似如下输出：

```
hello logstash
2015-05-15T03:34:30.111Z 0.0.0.0 hello logstash
```

在这个例子中，我们通过输入插件 stdin 和输出插件 stdout 来运行 Logstash，所以不管你输入什么，都会输出同样的信息。使用-e 参数可以在命令行中快速地测试配置是否有效。

接下来我们测试一下使用 codec 来美化输出的格式。先使用 *Ctrl+C* 命令终止正在运行的 Logstash，再以如下命令重新运行：

```
bin/logstash -e 'input { stdin { } } output { stdout { codec => rubydebug
} }'
```

现在再输入一些测试信息：

```
Hello PacktPub
  {
    "message" => " Hello PacktPub",
```

```
    "@timestamp" => "2015-05-20T23:48:05.335Z",
    "@version" => "1",
    "host" => "packtpub"
}
```

上面的输出信息是 Logstash 最常见的形式：

- "message"包含完整的输入信息或者事件。
- "@timestamp"包含了事件被索引时的时间。如果使用了日期过滤插件，也可能是 message 中的某个指定事件时间的字段。
- "host"通常来说表示事件发生的主机。

Logstash 的文件输入插件

Logstash 可以很容易地配置成读取日志文件作为输入源。

例如，可以读取 Apache 日志文件作为输入，然后输出到标准输出，配置如下：

```
input { file {
      type => "apache"
      path => "/user/packtpub/intro-to-elk/elk.log"
    }
} output {
    stdout {
      codec => rubydebug
} }
```

Logstash 的 Elasticsearch 输出插件

Logstash 可以被配置将所有的输入输出到一个 Elasticsearch 实例中。这在 ELK 平台中是最常用的配置：

```
bin/logstash -e 'input { stdin { } } output { elasticsearch { host
= localhost } }'
```

然后当然就是输入日志了。

你可以通过如下接口搜索 Elasticsearch 的索引：http://localhost:9200/_search。

配置 Logstash

Logstash 的配置文件使用的是 JSON 格式。根据为处理事件而添加的插件类型的不同，配置文件被分为不同的节。例如：

```
# 这是一条注释。你应该使用注释来描述配置文件中的各个部分
   input {
   ... }
filter {
   ...
}
output {
   ...
}
```

每一节都包含了一个或者多个插件的配置选项。如果使用了多个过滤插件，则配置文件中的顺序就指定了应用到事件处理流的顺序。在运行 `logstash` 时，可以通过`-flag` 参数指定配置文件的路径，甚至可以是一个包含多个不同类型如输入、过滤和输出插件的配置文件的目录：

```
bin/logstash -f ../conf/logstash.conf
```



如果希望在运行前测试配置文件的语法错误，可以执行如下命令：
`bin/logstash -configtest ../conf/logstash.conf`
上述命令只会检查配置文件，而不是真正地运行 `logstash`。

Logstash 在 JVM 中运行，需要消耗大量的资源。通常来说，Logstash 需要消耗大量的内存。很显然，在一台低配置的机器上需要采集日志而又不想影响到应用的性能时，这会是一个较大的挑战。

为了减少对资源的消耗，也可以使用 Logstash forworder 工具（以前叫 Lumberjack）。这个工具使用 Lumberjack 协议，可以安全地传输压缩后的日志，从而减少资源消耗以及降低对带宽的需求。这个工具要求输入只能是文件，输出则可以传送到多个目的地。

当然，发送日志也有其他的方法。比如在 Linux 机器上可以使用 `rsyslog`，在 Windows 上也有诸如 `nxlog` 和 `syslog-ng` 这样的工具。另外，还有一个轻量级的日志传输工具叫

Log-Courier (https://github. com/driskell/log-courier)，它是在 Logstash forworder 的基础上改进而来的。

安装 Logstash forworder

从官网的下载页面下载最新的 Logstash forworder 版本



最新的 Logstash forworder 发行版本地址在 https://www.elastic.co/downloads/logstash。

为了在采集器和接收服务器之间建立安全的通讯连接，需要准备一个配置文件，其中包含输入插件和 SSL 认证信息，并执行如下命令：

```
Logstash forwarder -config Logstash forwarder.conf
```

在 Logstash 中，我们可以使用 Lumberjack 插件来接收 forworder 发送来的数据：

```
input {
  lumberjack {
    # 监听的端口
    port => 12345
    port => 12345
    # ssl cert和key文件的存放路径
    ssl_certificate => "path/to/ssl.crt"
    ssl_key => "path/to/ssl.key"
    # 设置日志的类型
    type => "log type"
  }
```

Logstash 插件

Logstash 的常用插件有三类：

- 输入插件
- 过滤插件
- 输出插件

输入插件

常见的 Logstash 输入插件如下。

- **file**：从日志文件中读取事件流。
- **redis**：从 redis 实例中读取事件流。
- **stdin**：从标准输入中读取事件流。
- **syslog**：通过网络从 syslog 消息中读取事件流。
- **ganglia**：通过 udp 网络读取 ganglia 包中的事件流。
- **lumberjack**：使用 lumberjack 协议读取事件流。
- **eventlog**：从 Windows 事件日志中读取事件流。
- **s3**：从亚马逊 s3 存储桶的文件中读取事件流。
- **elasticsearch**：从 Elasticsearch 集群的搜索结果中读取事件流。

过滤插件

常见的 Logstash 过滤插件如下。

- **date**：从流入的事件中解析日期字段作为 Logstash 的 timestamp 字段，这对后面的分析非常有用。
- **drop**：从流入的事件中抛弃符合特定过滤条件的所有数据。
- **grok**：这是一个非常强大的过滤插件，可以将非结构化的日志事件解析成结构化的数据。
- **multiline**：可以将同一个输入源中的多行数据解析为一条日志事件。
- **dns**：该插件可以将任意指定的字段解析为 IP 地址。
- **mutate**：该插件可以重命名、删除、修改或者替换事件中的任意字段。
- **geoip**：根据 Maxmind IP 数据库，可以将 IP 字段解析出地理位置相关的信息。

输出插件

常见的 Logstash 输出插件如下。

- **file**：将事件写入到磁盘上的文件中。
- **e-mail**：在接收到输出时，根据某些特定的条件发送邮件。

- **elasticsearch**：将输出数据保存到 Elasticsearch 集群中，这是 Logstash 最常见也是推荐的输出方式。
- **stdout**：将事件写入标准输出。
- **redis**：将事件写入到 redis 的队列作为代理，在很多 ELK 部署中，这也是一种常见的方式。
- **mongodb**：将输出信息写入 mongodb。
- **kafka**：将事件写入 Kafka 的主题。

安装 Kibana

在安装和运行 Kibana 之前，需要满足一些前提条件：

- 确认 Elasticsearch 已经安装好，并且 HTTP 服务已经在 9200 端口（默认）上运行
- 确认 Kibana 已经配置好运行 Elasticsearch 的主机和端口号（参考后面的配置 Kibaba 一节）。

从官网的下载页面下载最新的 Kibana 版本。



最新的 Kibana 发行版本参见
https://www.elastic.co/downloads/kibana。

```
curl -O https://download.elastic.co/kibana/kibana/kibana-4.0.2-linux-x64.
tar.gz
```

然后在本地文件系统中解压 `kibana-4.0.2-linux-x64.tar.gz`，并且创建一个软链接。

```
tar -zxvf kibana-4.0.2-linux-x64.tar.gz

ln -s kibana-4.0.2-linux-x64 kibana
```

然后进入 `Kibana` 目录：

```
cd kibana
```

配置 Kibana

Kibana 的配置文件放置在 Kibana 安装路径下的 config 目录中：

```
config/kibana.yml
```

下面介绍 Kibana 的一些重要配置项。

控制使用的端口：

```
port: 5601.
```

指定绑定的主机：

```
host: "localhost".
```

设置 elasticsearch_url 指向 Elasticsearch 实例，默认是 localhost：

```
elasticsearch_url: http://localhost:9200
```

运行 Kibana

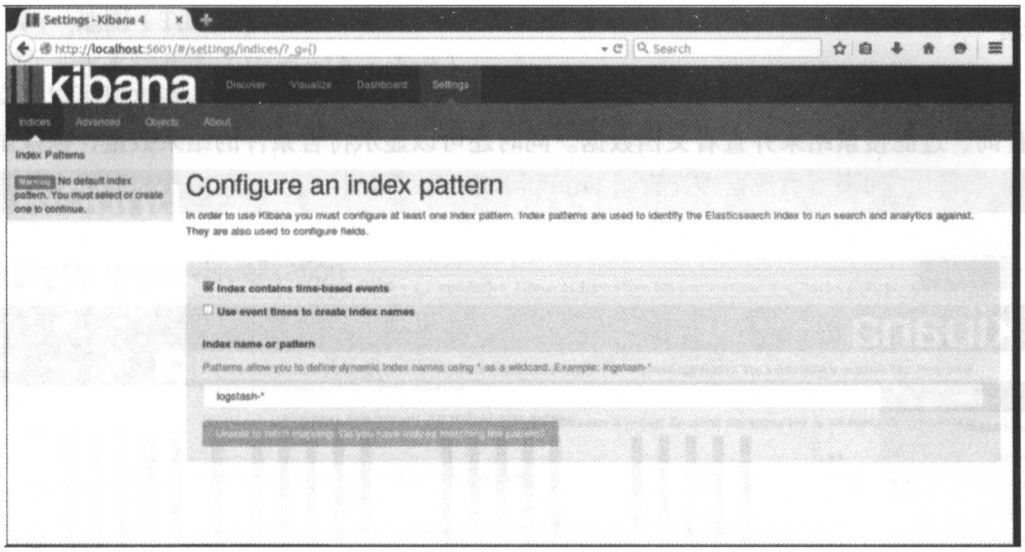
执行如下命令可手动启动 Kibana：

```
bin/kibana
```

在浏览器中打开如下链接，确认 Kibana 已经运行在 5601 端口：

```
http://localhost:5601
```

Kibana 的界面看起来应当如下：



Kibana 界面



为了显示 Elasticsearch 中索引的数据，还需要指定索引的名字或者模式。默认情况下，由于假定数据是通过 Logstash 存入 Elasticsearch 的，Kibana 预设的索引是 `logstash-*`。如果在 Logstash 的输出插件中修改过索引名，那么在 Kibana 的配置中也需要做同步修改。

对比 **Kibana 3** 与 **Kibana 4**

Kibana 4 是对 Kibana 3 的一个重大升级。Kibana 4 提供了一些高级工具，在可视化方面提供了更多的灵活性，可以使用 Elasticsearch 更多的高级特性。Kibana 3 必须安装在 Web 服务器上；Kibana 4 则可以作为独立应用。对比 Kibana 3，Kibana 4 的一些新特性包括：

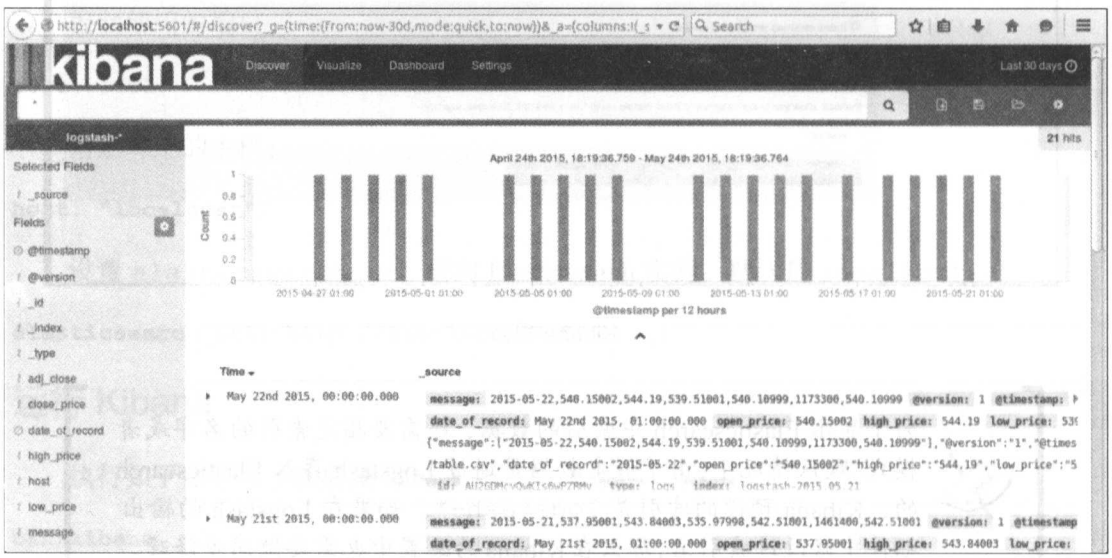
- 搜索结果高亮显示。
- 自带 Nodejs 后端作为 Web 服务器。
- 基于聚合的高级分析特性，如：唯一性统计、非日期类型的直方图、范围查询和百分比查询等。

Kibana 的界面

如前面显示的 Kibana 界面的截图所示，Kibana 的界面由四个主要的部件组成：**探索**、**可视化**、**仪表盘**和**设置**。

探索

探索（Discover）页可以交互式地查看符合所选的索引模式的数据。在这里可以提交搜索查询、过滤搜索结果并查看文档数据。同时还可以显示符合条件的结果数量，以及基于字段的统计。如果索引数据中配置了 timestamp 字段，默认还会显示基于时间的文档直方图。



Kibana 探索页

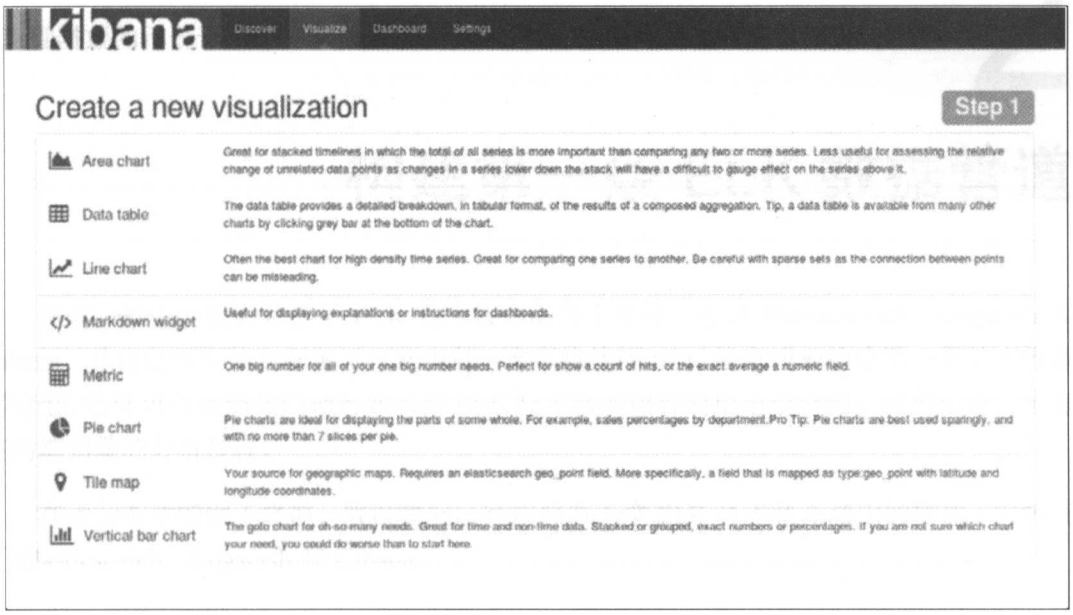
可视化

在**可视化**（Visualiza）页面中，可以基于不同的数据源，比如新的交换式搜索、已保存搜索，或者其他已经存在的可视化部件，来创建新的可视化部件。Kibana 4 还可以通过新的可视化向导来创建如下的部件：

- 区域图（Area chart）
- 数据表（Data table）
- 折线图（Linechart）
- Markdown 部件（Mardown widget）
- 度量（Metric）
- 饼图（Pie chart）

- 地图（Tile map）
- 水平柱状图（Vertical bar chart）

这些可视化部件可以保存、独立使用或者在仪表盘中使用。



Kibana 可视化页面

仪表盘

仪表盘（Dashboard）是保存在不同组的可视化部件的集合。这些可视化组件可以通过拖拽的方式自由布局，也可以根据数据的重要性来排序。仪表盘可以保存、分享或者在任意时间点载入。

设置

在**设置**（Settings）页面可以配置我们需要探索的 Elasticsearch 的索引名或者各种索引模式。在这里也可以看到所指定的索引模式中被索引的字段和字段类型，也可以创建脚本化的字段，这是一种基于索引数据计算而来的字段。

总结

在本章中，我们对 ELK 技术栈有了基本的了解，并且指出了为什么需要日志分析以及 ELK 技术栈的特别之处。另外，也说明了如何设置 Elasticwsearch、Logstash 和 Kibana。

在下一章中，我们将看到如何利用 ELK 技术栈快速地创建一个用于分析的数据管道。

2

构建第一条 ELK 数据管道

在前一章中，我们熟悉了 ELK 技术栈中的各个组件，包括 Elasticsearch、Logstash 和 Kibana，并且已经安装和配置了这些组件。本章将使用 ELK 技术栈来构建第一条基本的数据管道。这样可以帮助我们理解如何将 ELK 技术栈的组件简单地组合到一起来构建一个完整的端到端的分析过程。

在运行本章的例子之前，假设我们已经按照第 1 章 ELK 技术栈介绍中的描述，安装好了 Elasticsearch、Logstash 和 Kibana。

输入的数据集

在我们的例子中，将要使用的数据集是从 2014 年 7 月 1 号到 2014 年 12 月 31 号六个月内 Google（GOOG）每天的股票价格数据。这是一个很好的数据集，可以帮助我们了解如何利用 ELK 快速分析类似的简单数据集。



这个数据集可以便捷地从如下源地址下载到：http://finance.yahoo.com/q/hp?s=GOOG。

输入数据集的数据格式

这个数据集中最重要的字段包括 Date（日期）、Open Price（开盘价）、Close Price（收盘价）、High Price（最高价）、Volume（成交量）和 Adjusted Price（调整价格）。

下表显示了数据集中的部分示例数据。实际的数据集是 CSV 格式的。

Date	Open	High	Low	Close	Volume	Adj Close
Dec 31, 2014	531.25	532.60	525.80	526.40	1,368,200	526.40
Dec 30, 2014	528.09	531.15	527.13	530.42	876,300	530.42
Dec 29, 2014	532.19	535.48	530.01	530.33	2,278,500	530.33
Dec 26, 2014	528.77	534.25	527.31	534.03	1,036,000	534.03
Dec 24, 2014	530.51	531.76	527.02	528.77	705,900	528.77
Dec 23, 2014	527.00	534.56	526.29	530.59	2,197,600	530.59
Dec 22, 2014	516.08	526.46	516.08	524.87	2,723,800	524.87
Dec 19, 2014	511.51	517.72	506.91	516.35	3,690,200	516.35
Dec 18, 2014	512.95	513.87	504.70	511.10	2,926,700	511.10
Dec 17, 2014	497.00	507.00	496.81	504.89	2,883,200	504.89
Dec 16, 2014	511.56	513.05	489.00	495.39	3,964,300	495.39
Dec 15, 2014	522.74	523.10	513.27	513.80	2,813,400	513.80
Dec 12, 2014	523.51	528.50	518.66	518.66	1,994,600	518.66
Dec 11, 2014	527.80	533.92	527.10	528.34	1,610,800	528.34
Dec 10, 2014	533.08	536.33	525.56	526.06	1,712,300	526.06

我们需要把这些数据放到 ELK 技术栈能够访问的地方，以便于接下来的分析。

我们可以利用 Unix 的 head 命令来查看 CSV 文件的前面几条数据：

```
$ head GOOG.csv
2014-12-31,531.25244,532.60236,525.80237,526.4024,1368200,526.4024
2014-12-30,528.09241,531.1524,527.13239,530.42242,876300,530.42242
2014-12-29,532.19244,535.48242,530.01337,530.3324,2278500,530.3324
2014-12-26,528.7724,534.25244,527.31238,534.03247,1036000,534.03247
2014-12-24,530.51245,531.76141,527.0224,528.7724,705900,528.7724
2014-12-23,527.00238,534.56244,526.29236,530.59241,2197600,530.59241
2014-12-22,516.08234,526.4624,516.08234,524.87238,2723800,524.87238
2014-12-19,511.51233,517.72235,506.9133,516.35229,3690200,516.35229
2014-12-18,512.95233,513.87231,504.7023,511.10233,2926700,511.10233
```

每一行代表了某一天的股票价格数据，字段之间以逗号分割。

现在，我们已经对数据有所熟悉，接下来需要配置 ELK 技术栈，以便利用 Logstash 来解析和处理数据、在 Elasticsearch 中建立索引，并在 Kibana 中构建美观的可视化结果。

配置 Logstash 的输入

正如我们已知，Logstash 有非常丰富的各种类型的插件，包括输入、输出和过滤插件，它们可以根据需求来读取、解析和过滤每一行数据。我们将使用文件（file）输入插件读取源文件。

文件输入插件可以从文件中读取事件到输入流里，文件中的每一行会被当成一个事件处理。它能够自动识别和处理日志轮转。如果配置正确，它会维护读取位置并自动检测新的数据。它读取文件的方式就类似于：

```
tail -0f
```

通常来说，文件输入插件的配置看起来如下：

```
input {
  file {
    path => #字符串（文件的路径）(必选项)
    start_position => #字符串(可选项，默认为 "end")
    tags => #数组（可选项）
    type => #字符串（可选项）
}
}
```

- path：path 字段是文件输入插件唯一必填的配置项，表示需要的事件数据的源文件所在的路径。
- start_position：这个配置项定义了 Logstash 从源文件读取数据的开始位置，它的值可以是"beginning"或者"end"。默认是"end"，这样可以满足读取活动的流数据的场景需求。如果需要读取历史数据，可以设置为"beginning"。
- tags：可以是任意数量的字符串数组，能在随后基于 tags 来针对事件做一些过滤和处理。
- type：用来标记事件的特定类型，可以在随后的过滤和搜索中有所帮助。Type 字段会保存在 Elasticsearch 的文档中，并通过 Kibana 的_type 字段来进行展现。例如，我们可以将 type 设置为"error_log"或者"info_logs"。

接下来对我们的数据集配置如下的 Logstash：

```
input{
file{
```

```
path =>"/opt/logstash/input/GOOG.csv"
start_position =>"beginning"
}
}
```

我们在 path 属性中设置了 CSV 文件的路径，因为是历史数据，所以需要设置 start_position 为"beginning"。

过滤和处理输入数据

配置好输入文件后，接下来可以根据需要对输入数据进行过滤，以便识别出需要的字段并进行处理，以达到分析的目的。

过滤插件将对输入事件进行中间处理。我们可以对某些特定的列做条件过滤。

因为我们的输入文件是 CSV 文件，所以可以使用 csv 过滤插件。csv 过滤器可以对 CSV 格式的数据提取事件的字段进行解析并独立存储。除了逗号之外，它还可以解析处理任意使用其他分隔符的数据。一个典型的 csv 过滤插件的配置如下：

```
filter {
    csv {
        columns => #字段名数组.
        separator => #字符串 ; 默认值 - ","
    }
}
```

其中，columns 属性指定了 CSV 文件中的字段的名字，这是可选项。默认会将字段命名为 column 1、column 2 等等。

Separator 属性定义了输入文件中用来分割不同字段的分割符。默认的分隔符是逗号，但也可以是其他任意的分割符。

在我们的例子中，我们可以简单地配置 csv 过滤器如下：

```
filter {
    csv {
    columns =>
["date_of_record","open","high","low","close","volume","adj_close"
```

```
]
        separator => ","
    }
}
```

在这里，我们指定了 CSV 文件中的字段名，并且显式地定义了分割符是逗号。

现在，csv 过滤器配置好了，但还需要进一步的中间处理，来关联每个字段的数据类型。

首先，我们需要指定哪个列代表日期字段，以便它可以被显式地索引为日期类型，这样可以用于基于日期的过滤。Logstash 中有一个叫 date 的过滤器可以完成上述任务。一个典型的 date 过滤器配置如下：

```
filter {
  date {
    match => # 数组 (可选项), 默认值: []
    target => # 字符串 (可选项), 默认值: "@timestamp"
    timezone => # 字符串 (可选项)
}
}
```

其中，match 属性是一个[域，格式]格式的数组。这个格式为每个字段提供了一组可以应用于该字段的日期格式。例如，如果我们的日志文件有多种日期格式，可以配置如下：

```
match => [ "date_field", "MMM dd YYY HH:mm:ss",
          "MMM  d YYY HH:mm:ss",  "MMddYYYY","ISO8601" ]
```



Logstash 中的日期格式：允许使用的日期格式是根据 JodaTime 库的 DateTimeFormat 来决定的：http://joda-time.sourceforge.net/apidocs/org/joda/time/format/DateTimeFormat.html。

按照我们数据中的日期格式，date 过滤器可以配置如下：

```
date{
match => ["date_of_record", "yyyy-MM-dd"]
target => "@timestamp"
}
```

其中的 taget 过滤器定义了时间字段将映射到哪里。默认会映射到@timestamp 字段（这个字段代表时间的时间戳，默认是事件捕获时的时间）。在我们的案例中，因为我们采用了历史数据，不希望使用时间捕获时的时间作为@timestamp，而是使用记录生成时的时间，所以我们将 date 字段映射为@timestamp。这不是强制的，但建议这样做。

在修改了 date 字段的数据类型之后，下一步要做的是修改其他字段的数据类型，尤其是数字类型的比较或者其他操作。默认情况下，值的类型都是字符串。我们需要将其转换为整数，这样后续才能对数据进行聚合、比较的操作。

我们可以使用 mutate 过滤器将字段转换为指定的数据类型。这个过滤器可以用于对字段做各种常见的修改，包括修改数据类型、重命名、替换和删除字段。另外也可以用来合并两个字段、转换大小写、拆分字段等等。

一个典型的 mutate 过滤器看起来如下：

```
filter {
  mutate {
    convert => # 列以及数据类型的Hash值（可选项）
    join => # 用于关联的列的Hash值（可选项）
    lowercase => # 用于转换的字段数组（可选项）
    merge => # 用于合并的字段的Hash值（可选项）
    rename => # 原始和重命名字段的Hash值（可选项）
    replace => # 用于替换的字段的Hash值（可选项）
    split => # 用于分裂的字段的Hash值（可选项
    strip => # 字段数组（可选项）
    uppercase => # 字段数组（可选项）
}
}
```

来看一下我们的例子中 mutate 过滤器的实际配置：

```
mutate {
  convert => ["open","float"]
  convert => ["high ","float"]
  convert => ["low ","float"]
  convert => ["close ","float"]
  convert => ["volume","integer"]
```

```
    convert => ["adj_close","float"]
  }
```

我们使用了 convert 功能来将价格和成交量字段转换为浮点数和整数类型。有效的数据类型包括"integer"（整数）、"float"（浮点数）和"string"（字符串）。

将数据存储到 Elasticsearch

现在，我们已经配置好使用 Logstash 的 CSV 过滤器（用来处理数据），并且已根据数据类型对数据进行解析和处理。接下来需要将处理后的数据存储到 Elaticsearch，以便对不同字段做索引，这样后续就可以使用 Kibana 来展现。

我们将使用 Logstash 的 elasticsearch 输出插件来完成这个任务。

一个典型的 elasticsearch 输出插件的配置看起来如下：

```
output {
  elasticsearch {
    action => # 字符串（可选项），默认值："index"
    cluster => # 字符串（可选项）
    host => # 字符串（可选项）
    document_id => # 字符串（可选项），默认值：nil
    index => # 字符串（可选项），默认值："logstash-%{+YYYY.MM.dd}"
    index_type => # 字符串（可选项）
    port => # 字符串（可选项）
    protocol => # 字符串，取值为 ["node", "transport", "http"] 其中之一
（可选项）}
  }
}
```

- action：指定了对输入的文档执行的操作类型。默认是"index"（索引）。可用的操作包括"index"（索引）和"delete"（删除）。"index"操作会对文档进行索引，而"delete"操作则会根据文档 ID 删除文档。
- cluster：elasticsearch 中设置的集群名字。
- host：elasticsearch 的主机名或者 IP 地址。
- document_id：索引的文档 ID，对于删除文档或者覆盖已有的文档非常有用。

- index：指定了输入的事件需要写入的索引名字。默认情况下，数据按天索引，并且命名为"logstash-%{+YYYY.MM.dd}"。
- index_type：指定了事件写入的索引类型。该选项可以确保将相同类型的事件写入相同类型的索引。
- port：指定了 elasticsearch 服务使用的端口。
- protocol：连接到 Elasticsearch 的协议类型，可用的协议类型包括"http"、"node"和"transport"。

接下来看一下我们例子中 elaticsearch 输出插件的配置：

```
output{
elasticsearch {
}
}
```

对于索引和大多数其他设置，我们都采用了默认值。

上面我们分别对每个不同的插件配置单独做了说明，最后来看一下完整的 Logstash 配置是什么样子：

```
input{
file{

path =>"/opt/logstash/input/GOOG.csv"
start_position =>"beginning"

}

}

filter{
csv{

columns =>
["date_of_record","open","high","low","close","volume","adj_close"]

separator => ","
}

date {
```

```
match => ["date_of_record","yyyy-MM-dd"]
}
mutate {
convert => ["open","float"]
convert => ["high","float"]
convert => ["low","float"]
convert => ["close","float"]
convert => ["volume","integer"]
convert => ["adj_close","float"]
}
}
output{
elasticsearch {
host => "localhost"
}
}
```

我们将这个配置文件保存到 Logstash 的安装目录，命名为 `logstash.conf`。就像之前已经见过的，可以使用如下命令来运行 Logstash 并使用该配置文件：



在使用该配置文件运行 Logstash 之前，请确保 Elasticsearch 已经按照前一章的说明运行了。

```
$ bin/logstash -f logstash.conf
```

Logstash 将按照定义好的配置启动，并且持续地将输入的事件保存到 `elasticsearch` 的索引中。你可以在控制台中看到类似下面这样的输出信息：

```
May 31, 2015 4:04:54 PM org.elasticsearch.node.internal.InternalNode
```

```
start
INFO: [logstash-4004-9716] started
Logstash startup completed
```

到了这一步，我们可以打开 elasticsearch Kopf 插件的控制台，进一步确认是否已经有文档索引成功，并且可以对这些文档进行查询。



Elasticsearch Kopf 界面

如上图显示，已经有 129 个文档索引成功，可以确认我们的 Logstash 配置能够正常工作了。

使用 Kibana 可视化

现在我们已经验证数据在 Elasticsearch 中索引成功，接下来可以进一步看看 Kibana 界面，以便从数据中得到一些有用的分析。

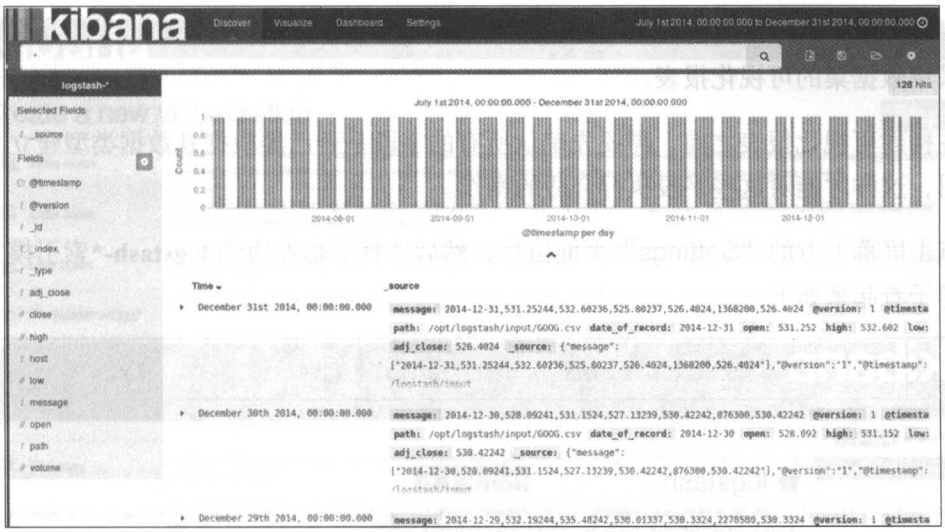
运行 Kibana

如前一章所述，我们将在 Kibana 安装目录中启动 Kibana 服务。

```
$ bin/kibana
```

然后可以打开浏览器，进入如下 URL 来查看 Kibana 启动和运行的情况，屏幕截图如下：

```
http://localhost:5601
```



Kibana 搜索界面

我们已经将 Kibana 设置为默认使用 **logstash-***索引，它会显示已索引数据的直方图统计，并且将相关数据以 JSON 格式的方式呈现。

首先，我们需要在数据的日期范围内设置日期过滤器，以构建我们的分析。因为使用的是从 2014 年 7 月 1 日到 2014 年 12 月 31 日的数据，日期过滤器也需要按照相同的日期范围来设置。

点击右上角的**时间过滤器**（Time Filter）图标，我们可以根据数据的日期范围来设置绝对**时间过滤器**，如下图所示：

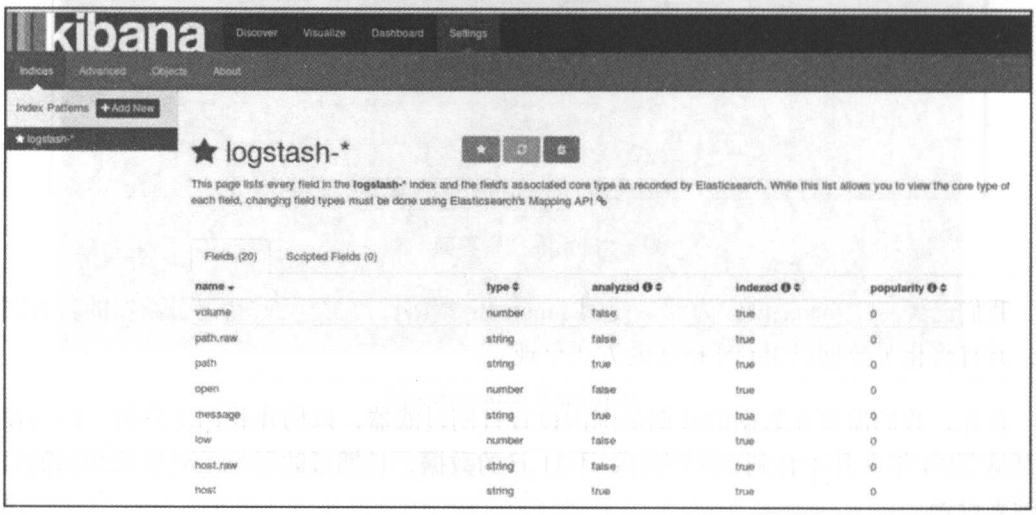


Kibana 时间过滤器

现在，我们已经完成设置，可以使用 Kibana 提供的丰富的可视化特性来构建美观的基于已采集数据集的可视化报表。

在构建可视化报表之前，需要先确认所有的字段是否已经根据其数据类型建立了正确的索引，这样才能对这些字段执行合适的操作。

点击屏幕上方的“Settings”页面链接，然后选择屏幕左边的 **logstash-*索引模式。页面的样子看起来如下：



Kibana 设置页面

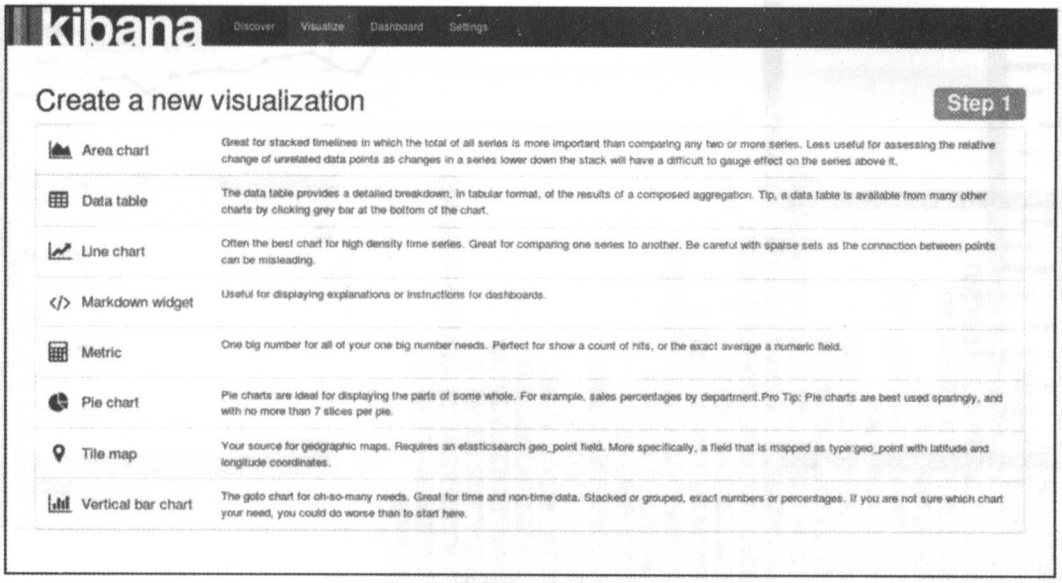
这里显示了所有已经索引的字段，以及它们的数据类型、索引状态和人气值。

Kibana 可视化组件

让我们先在 Kibana 可视化页面构建一些基本的可视化组件，稍后会在仪表盘中使用这些组件。

在 Kibana 主页上方点击可视化（Visualize）页面链接，然后点击新建可视化的图标。

此页显示多种可视化组件都可以用 Kibana 接口来实现：

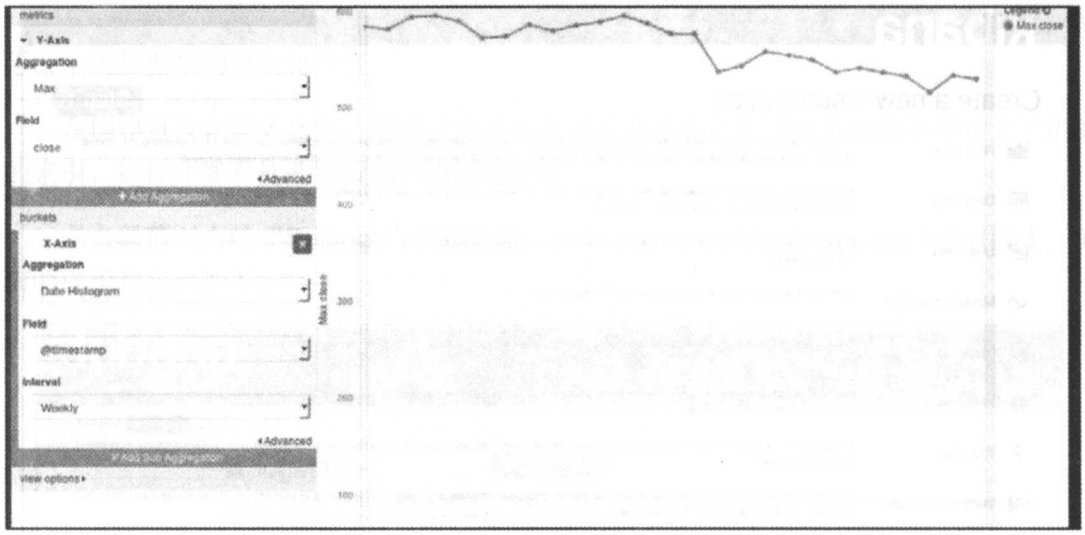


Kibana 可视化菜单

构建折线图

我们首先构建的可视化部件是一个折线图，用来显示六个月来 GOOG 每周收盘价的指数趋势。

从上图的可视化菜单中选择**折线图**，然后选择 **Y 轴**（Y-Axis）的聚合函数为 **Max**，**字段**为 **close**。在**桶**（buckets）的区域，选择**聚合**（Aggregation）为基于**@timestamp 字段**的**日期直方图**（Date Histogram），**间隔**（Interval）则选择**每周**（Weekly），然后点击**应用**（Apply）。



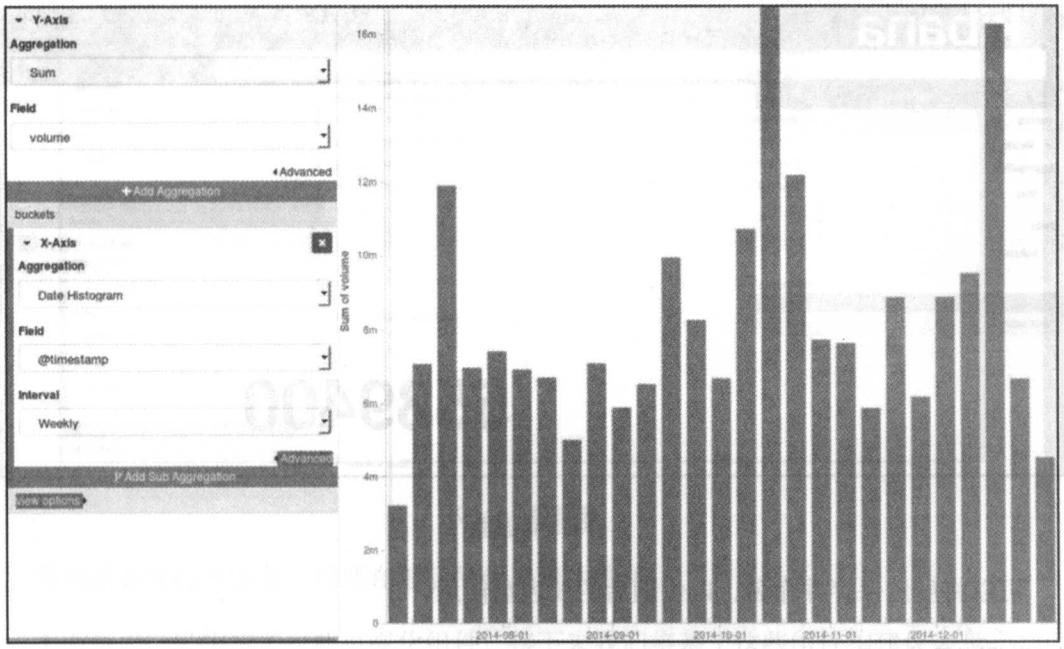
Kibana 折线图

现在，可以将上述折线图保存并命名，随后就可以在代表盘中使用。

构建柱状图

我们将构建一个垂直柱状图来呈现六个月内的成交量变化趋势。

在可视化菜单中选择**垂直柱状图**，选择 Y 轴的**聚合函数**为 Sum，**字段**为 volume。在**桶**的区域，选择 X 轴的**聚合函数**为基于@timestamp **字段**的**日期直方图**，**间隔**则选择**每周**，然后点击**应用**，就可以看到六个月内的成交量趋势的柱状图。



Kibana 垂直柱状图

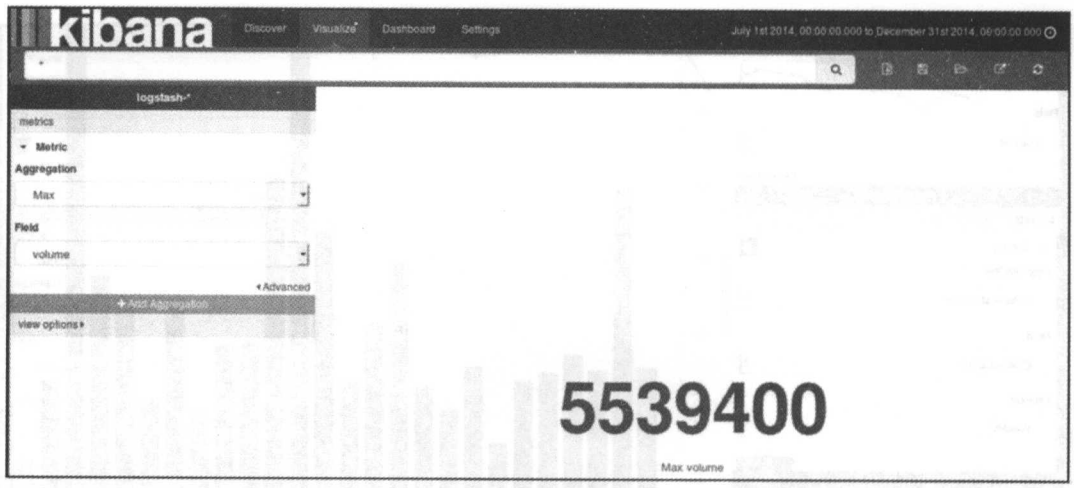
现在，可以将上述柱状图保存并命名，随后就可以在仪表盘中使用。

构建度量

当我们想展示数据某方面的独特性时，大多用度量。

我们将用度量来展示过去六个月中每天的**单日最高交易量**。

点击可视化菜单中的度量，选择**度量的聚合函数**为 Max，**字段**为 volume，然后点击**应用**，就可以在右边看到可视化的结果，如下图所示：



Kibana 度量

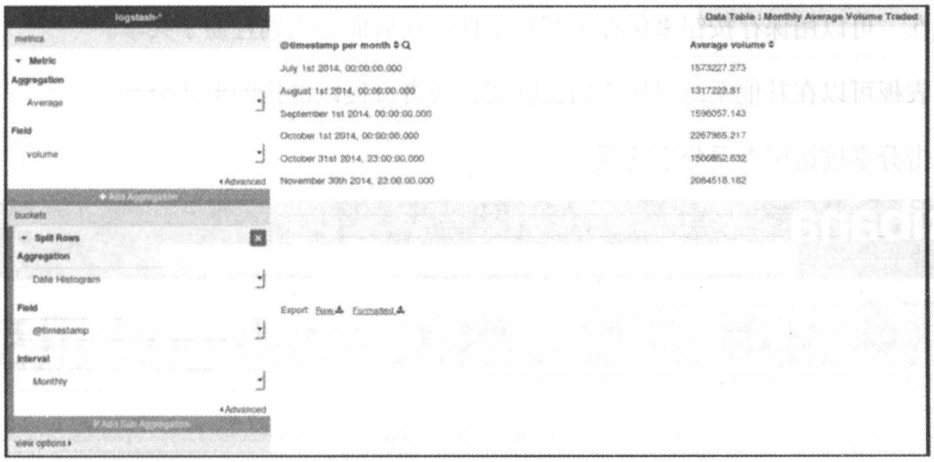
可以将**度量**保存并命名，以供随后的仪表盘使用。

构建数据表

数据表可以以表格的形式来显示某些组合聚合结果的详细数据。

接下来我们将创建一个六个月内的**月度平均**成交量的数据表。

选择可视化菜单中的**数据表**，点击**拆分行**（split rows），选择度量值的**聚合函数**为**求平均值**（Average），**字段**为 volume。在**桶**的区域，选择**聚合函数**为基于**@timestamp 字段**的**日期直方图**，**间隔**为月度（Monthly）。点击**应用**就可以看到如下图的信息：



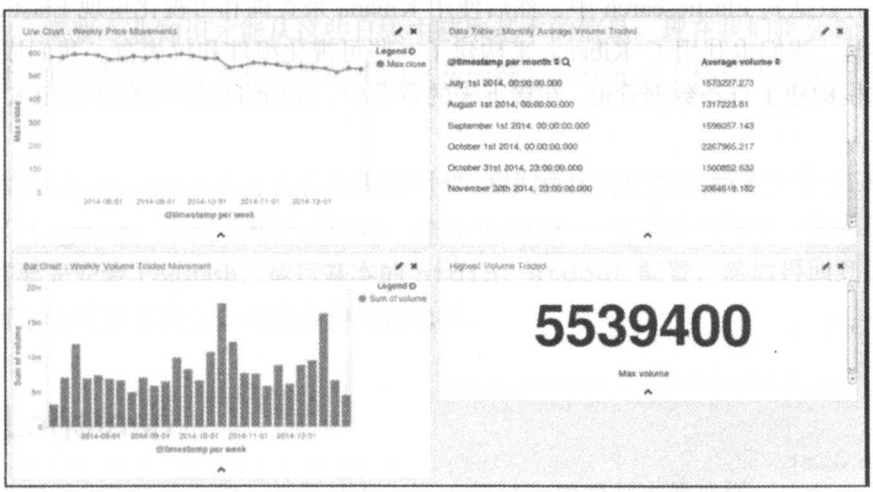
Kibana 数据表

将数据表保持并命名，以便在随后的仪表盘中使用。

上面我们已经构建了一些可视化组件，接下来我们将基于这些组件构建仪表盘。

在页面的顶部选择仪表盘页面链接，点击**增加可视化**（Add Visualization）链接，选择已保存的可视化组件并在仪表盘页面上进行布局。

下图是一个包含了折线图、柱状图、数据表和度量值的仪表盘：

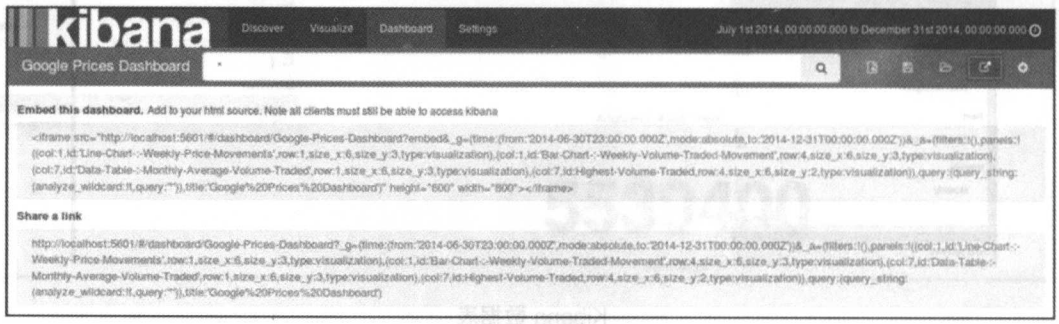


Kibana 仪表盘

现在，可以用保存按钮来保存可视化组件，在后面会用到且易于共享。

仪表板可以在其他系统里作为内嵌框架，或者直接以链接的形式分享。

点击分享按钮可查看分享选项：



Kibana 分享选项

到这里，如果你已经完成了上面所有的操作，那么你的首条 ELK 数据管道就设置成功了。

总结

在本章中，我们展示了如何在 Logstash 中利用不同的输入、过滤和输出插件来收集、解析和索引数据到 Elasticsearch 中，然后使用 Kibana 来查询和可视化呈现 Elasticsearch 中的索引数据。我们也利用了 Kibana 来构建了一些可视化组件和仪表盘。我们成功地使用 ELK 技术栈构建了首条数据管道。在接下来的章节中，我们将更详细地讨论 ELK 的每个独立的组件。

3

使用 Logstash 采集、解析和转换数据

现在，我们应当对 ELK 技术栈和它在日志分析或事件分析系统中扮演的角色有了基本的理解。在第 2 章 *构建第一条 ELK 数据管道*中，我们使用 ELK 技术栈构建了基于 GOOG 股票价格数据的分析系统，理解了 ELK 技术栈中各个组件在数据管道中的作用。

在本章中，我们将介绍 ELK 技术栈中最重要的组件——Logstash 的更多细节，理解 Logstash 如何采集、解析并将各种格式和类型的数据转换成通用格式，然后被用来为不同的应用构建多样的分析系统。

在第 1 章 ELK 技术栈介绍中已经了解日志分析的重要性并发现了日志分析方法存在的问题。日志中的日志格式、日期和时间的格式也多种多样。通常，每个应用有各自特定格式的日志，需要专业知识才能从这些日志中获取基本的信息。现在我们将会看到 Logstash 如何通过丰富多样的插件来帮助我们克服这些问题，并在此之上构建可简单配置、管理的可视化系统。

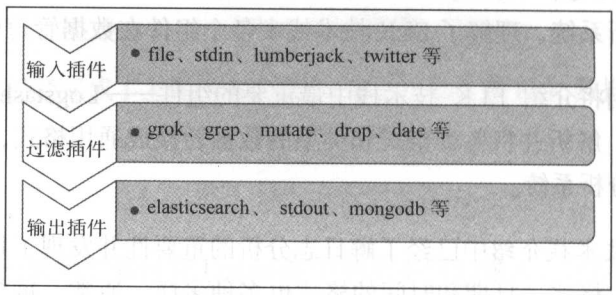
在本章讲解 Logstash 插件各种配置的时候，我们假定你已经按照第 1 章 ELK 技术栈介绍安装了 Logstash，完成了基础的配置，并且可以运行一些简单的例子。如果还没有这么做，你需要先安装 Logstash，运行基本的 `stdin`、`stdout` 配置，然后再回到这一章，这样你就可以尝试本章将会介绍的各种配置选项。

配置 Logstash

如我们在前几章的介绍，常见的 Logstash 插件配置看起来像这样：

```
input {
}
filter {
}
output {
}
```

Logstash 配置包括一系列输入插件、过滤插件和输出插件以及这些插件各自的属性。每一个插件在解析、处理并最终将数据以特定的格式输出的过程中都分别扮演了重要的角色。输入插件将源头数据转换成通用格式的事件，过滤插件修改这些事件，最终输出插件将它们输出到其他系统。



Logstash 插件

Logstash 插件

Logstash 拥有一系列插件，可以帮助各种输入和输出的数据源进行整合。我们来探讨各种现有的插件。

列出 Logstash 的所有插件

使用下面的命令可以列出当前 Logstash 安装版本中的所有可用插件：

```
bin/plugin list
```

使用下面的命令可以列出插件名中包含特定模式的插件：

```
bin/plugin list <namefragment>
```

使用下面的命令可以列出指定分组的插件[1]

```
bin/plugin list --group <group name>
bin/plugin list --group output
```

在探讨各种插件配置之前，我们先了解一下插件配置属性的数据类型和条件表达式。

插件属性的数据类型

每个 Logstash 插件都要求设置一些配置项或属性。这些属性的值可以是下面这些重要数据类型之一。

数组（Array）

属性的数组是值的集合。

下面是数组的例子：

```
path => ["value1", "value2""]
```



=>符号是 Logstash 配置中给插件属性赋值的操作符。

布尔值（Boolean）

布尔值只能是 `true` 或 `false`（注意没有引号）。

下面是一个布尔值的例子：

```
periodic_flush => false
```

编解码器（Codec）

编解码器实际上并不是一种数据类型，它是在输入或输出的时候对数据进行解码或编码的一种方式。

注1 命令中 group name 可以取 input、output、filter 或 codec 这几种值——译者注

下面是一个编解码器的例子：

```
codec =>"json"
```

上面这个例子指定在输出的时候，编解码器会将所有输出数据编码成 JSON 格式。

哈希（Hash）

哈希是由一系列键值对组成的集合。它以"key"=>"value"这种形式表示，多个值用空格分割。

下面是一个哈希的例子：

```
match => {
"key1"=>"value1" "key2"=>"value2"}
```

字符串（String）

字符串是使用引号包含起来的字符序列。

下面是一个字符串的例子：

```
value => "Welcome to ELK"
```

注释（Comment）

注释以字符#开头。

下面是一个注释的例子：

```
# 这是一个注释
```

字段引用

字段可以使用[field_name]的方式引用，嵌套字段可以使用[level1][level2]的方式指定。

Logstash 条件语句

在某些条件下 Logstash 可以用条件语句来过滤事件或日志记录。Logstash 中的条件处

理和其他编程语言中的类似，使用 if,if else 和 else 语句。多个 if else 语句块可以嵌套。

条件语句的语法如下：

```
if <conditional expression1> {
# 一些处理语句
}
else if <conditional expression2> {
# 一些处理语句
}
else {
# 一些其他语句
}
```

条件语句可以与比较运算符、逻辑运算符和单目运算符一起使用。

比较运算符包括以下几种。

- **相等运算符**：==, !=, <, >, <=, >=
- **正则表达式**：=~, !~
- **包含**：in, not in
- 逻辑运算符包括 and, or, nand, xor
- 单目运算符包括!

我们来看一个例子：

```
filter {
  if [action] == "login"] {
    mutate { remove => "password"}
  }
}
```

多个表达式可以使用逻辑运算符连接起来。

下面是一个例子：

```
output {
  # 当产品出错时发送邮件
```

```
    if [loglevel] == "ERROR"and [deployment] == "production"{
     email {

       }

    }
}
```

Logstash 插件的类型

下面是 Logstash 插件的类型：

- 输入（Input）
- 过滤器（Filter）
- 输出（Output）
- 编解码（Codec）

接下来我们来了解一些在构建大多数日志分析数据管道场景下会用到的最重要的输入、输出、过滤和编解码插件。

输入插件

输入插件可用于将一组事件传递到 Logstash。一些重要的输入插件如下面所列。

文件（file）

文件输入插件用于将事件和日志文件输入给 Logstash。它会自动检测日志文件轮转，并从上次读取的位点读取数据。



Logstash 文件输入插件将文件读取的最新位点保存在称为 sincdb 的文件中。Sincedb 文件默认路径是$HOME/.sincdb*。文件的路径和刷新频率可以通过 sincedb_path 和 sincdb_write_interval 这两个配置项设置。

文件输入插件最基本的配置如下：

```
input {
file {
path=> "/path/to/logfiles"
}
```

唯一的必选属性是文件的路径 *path*。我们来看一下如何利用文件输入插件的配置属性来读取不同类型的文件。

配置选项

文件输入插件有下面这些配置选项。

add_field

用于为传入事件增加字段、哈希类型，默认值是 { }。

下面是一个例子：

```
add_field => {"input_time"=> "%{@timestamp}"}
```

codec

用于指定编解码器中解码特定类型的输入。例如：`codec =>"json"`用于解码 `json` 输入。Codec 的默认值是`"plain"`。

delimiter

用于指定行分隔符，默认值是`"\n"`。

exclude

用于排除指定类型的文件，数组类型是数组（`array`）。

下面是一个例子：

```
path => ["/app/packtpub/logs/*"]
exclude => "*.gz"
```

这个配置将所有 gzip 文件排除在外。

Path

这是文件输入插件的唯一一个必选属性。它以数组的形式指定了需要读取的日志文件的路径。

Sincedb_path

指定 sincedb 文件的路径，记录被监视文件的当前读取位点，默认值是 $HOME/.sincedb*。

sincedb_write_interval

指定 sincedb 文件的写入频率（单位是秒），默认是 15 秒。

start_position

这个属性有两个可选值："beginning"和"end"。它指定输入文件的初始读取位点。默认是"end"，大多数情况下用于实时流数据。如果需要读取历史数据，可以设置成"beginning"。



这个选项只在文件第一次读取的时候起作用，称为“初次接触”。读取位点在 sincedb 文件中。后续读取时，这个参数不起作用，除非将 sincedb 文件删除。

tags

这个参数用于给输入事件增加一列标签。给事件增加标签有助于后续对事件使用条件语句进行分析。通常将一些数据打上“已处理”的标签非常有用，还可以使用这些标签来决定后续的处理。

例如，如果我们在标签中指定“已处理”：

```
tags => ["processed"]
```

可以在过滤器中使用条件判断：

```
filter {
if "processed" in tags[ ] {

}
}
```

type

这个选项在使用 Logstash 处理多种不同类型的输入流时非常有用。你可以给多个输入路径中配置的不同类型的事件指定 type 名称，后续就可以使用它们单独过滤并处理。

下面是一个例子：

```
input{
file {
path => ["var/log/syslog/*"]
type => "syslog"
}
file {
path => ["var/log/apache/*"]
type => "apache"
}
}
```

在过滤器中，我们可以基于 type 处理日志：

```
filter {
if [type] == "syslog"{
grok{

}

}
if [type] == "apache"{
grok{
}
}
}
```

在前面的例子中，我们为输入文件配置了不同的 type 属性："syslog"和"apache"。后续在过滤日志流的时候，可以基于 type 字段使用条件判断来过滤日志。

标准输入（stdin）

stdin 输入插件用于从标准输入中读取事件和日志。

基本的配置看起来如下：

```
stdin {

}
```

当我们这样配置 stdin 插件的时候，终端上的任何输入都会作为 Logstash 日志管道的输入。这通常用来在真实的文件或事件输入前进行第一层配置测试。

配置选项

stdin 输入插件有下面这些选项可以配置：

`add_field`

stdin 插件的 add_field 配置和 file 插件的一样且用途类似。

codec

用于在将数据传递给数据管道之前对输入数据进行解码，默认值是"line"。

tags

stdin 插件的 tags 配置和 file 插件的一样且用途类似。

type

stdin 插件的 type 配置和 file 插件的一样且用途类似。

twitter

出于不同的目的，你可能需要对 Twitter 数据流中基于不同兴趣的话题做分析，比如情感分析、话题的趋势分析等等。twitter 插件对于从 Twitter 流 API 读取事件很有帮助。使用这个插件需要提供消费者 key、消费者秘钥、关键词、oauth 口令和 oauth 口令秘钥。

这些详细信息可以通过在 twiiter 开发者 API 页面（https://dev.twitter. com/apps/new）上注册一个应用来获取：

```
twitter {
    consumer_key => "你的消费者key"
    keywords => "你用于过滤数据流的关键词"
    consumer_secret => "你的消费者秘钥"
    oauth_token => "你的oauth口令"
    oauth_token_secret => "你的oauth口令秘钥"
  }
```

配置选项

twitter 插件有下面这些配置选项可用。

add_field

twitter 插件的 add_field 配置和 file 插件的一样且用途类似。

Codec

twitter 插件的 codec 配置和 file 插件的一样且用途类似。

consumer_key

这是一个必选的配置项，没有默认值。具体值可以从 Twitter app 注册页面获取，值的类型是字符串。

consumer_secret

这是一个必选的配置项，没有默认值。具体值可以从 Twitter app 注册页面获取，值的类型是字符串。

full_tweet

这是一个布尔类型的配置项，默认值是 false。用于指示是否完整的记录使用 Twitter 流 API 获取的 tweet 对象。

keywords

这是一个数组类型的必选参数，没有默认值。用于指定一组关键词跟踪 Twitter 流。

下面是一个例子：

```
keywords => ["elk", "packtpub"]
```

oauth_token

oauth_token 选项也是从 Twitter 开发者 API 页面获取的。



在获取到你的消费者 key 和消费者秘钥之后，点击“创建我的访问口令”可以创建 oauth 口令和 oauth 口令秘钥。

oauth_token_secret

这个选项的值也是从 Twitter 开发者 API 页面获取的。

tags

twitter 插件的 tags 配置项和 file 插件的一样且用途类似。

type

twitter 插件的 type 配置项和 file 插件的一样且用途类似。

lumberjack

lumberjack 输入插件可用于从使用 lumberjack 协议的 Logstash 转发器中接收事件。

lmberjack 插件的基本必选配置项如下：

```
lumberjack {
    port =>
    ssl_certificate =>
    ssl_key =>
}
```



Lumberjack 或者说 Logstash 转发器，是一个轻量级的日志转发器，用于从源端系统转发日志事件。Logstash 进程消耗的内存比较多，不推荐在每个需要上传日志的节点上安装 Logstash。Logstash 转发器是一个轻量级版本的 Logstash，提供了低延迟、安全可靠的日志传输，同时消耗的资源也更低。

关于 Lumberjack 或 Logstash forwarder 的更详细的信息可以从这里获取：https://github.com/elastic/logtash-forwarder。

配置选项

`lumberjack` 输入插件有下面这些配置选项可用。

add_field

`lumberjack` 插件的 `add_field` 配置项和 `file` 插件的一样且用途类似。

Codec

`lumberjack` 插件的 `codec` 配置项和 `file` 插件的一样且用途类似。

Host

指定监听的主机，默认值是"`0.0.0.0`"。

port

这是一个必选的数字类型的配置项，用于指定监听端口。没有默认值。

ssl_certificate

指定连接用的 SSL 证书路径。这是一个必选参数。

下面是一个例子：

```
ssl_certificate => "/etc/ssl/logstash.pub"
```

ssl_key

指定用于连接的 SSL key 路径，是必选参数。

下面是一个例子：

```
ssl_key => "/etc/ssl/logstash.key"
```

ssl_key_passphrase

指定用于连接的 SSL key 的秘钥。

tags

lumberjack 输入插件的 tags 配置项和 file 输入插件的一样且用途类似。

type

lumberjack 输入插件的 type 配置项和 file 输入插件的一样且用途类似。

redis

redis 插件用于从 redis 实例中读取事件和日志。



在 ELK 技术栈中，Redis 经常用作 Logstash forwarder 输入数据的消息代理，将输入数据缓存到队列，等待索引器读取日志。这有助于保持系统在高压力下正常运行。

redis 输入插件基本的配置如下：

```
redis {
}
```

配置选项

redis 输入插件有下面这些配置选项可以使用。

add_field

redis 插件的 add_field 配置项和 file 插件的一样且用途类似。

codec

redis 插件的 codec 配置项和 file 插件的一样且用途类似。

data_type

data_type 选项可以配置为"list", "channel"或"pattern_channel"。

Logstash redis 输入插件的官方文档中描述(https://www.elastic.co/guide/en/logstash/current/plugins-inputs-redis.html):

"如果 redis_type 配置为 list，我们会对 key 进行 BLPOP 操作。如果 redis_type 是 channel，我们会使用 SUBSCRIBE 命令订阅 key。如果 redis_types 是 pattern_channel，我们会使用 PSUBSCRIBE 命令订阅 key"。



在消费端和发布端都使用 redis 时，key 和 data_type 在两端必须设置相同。

host

指定 redis 服务器的主机名，默认值是"127.0.0.1"。

key

指定 redis 的 key，包括"list"或者"channel"。

password

这是一个密码类型的配置项，指定用于连接的密码。

port

指定 redis 实例的端口，默认值是 6379。

所有可用 Logstash 输入插件的详细列表、最新文档可以到这里查看：https://www.elastic.co/guide/en/logstash/current/input-plugins.html。

到目前为止我们已经了解了一些 Logstash 的重要输入插件，下面我们来了解一些输出插件。

输出插件

Logstash 提供了多种输出插件，可以将输入事件和几乎所有类型的输出目标进行整合。我们来仔细学习一些最常用的输出插件。

csv

csv 输出插件用于把输出写入 CSV 文件，需要指定写入的字段和文件的路径。

csv 输出插件的基本配置如下：

```
csv {
    fields => ["date", "open_price", "close_price"]
    path => "/path/to/file.csv"
}
```

配置选项

csv 输出插件有下面这些配置选项可用。

codec

指定将数据从 Logstash 输出时的编码方式。默认值为"plain"，将数据原样输出。

csv_options

csv_options 选项用于给 csv 输出指定高级选项，包括改变默认列分隔符和行分隔符。

下面是一个例子：

```
csv_options => {"col_sep" => "\t" "row_sep" => "\r\n"}
```

fields

fields 是一个必选参数，用于指定 CSV 输出文件的字段。它指定为一个有字段名的数组，其输出文件中的字段顺序和参数配置数组中的顺序保持一致。这个参数没有默认值。

gzip

这是一个布尔类型的参数，用于指定是否以 gzip 格式压缩输出文件。默认值是 false。

path

path 是一个必选参数，用于指定输出 CSV 文件的路径。

文件

文件输出插件和文件输入插件有点类似，用于将事件写入文件。

文件输出插件的基本配置如下：

```
file
{
path => "/path/to/file"
}
```

配置选项

文件输出插件有这些配置项可用：

- codec
- gzip
- max_size
- path

这些选项大部分之前都已经讨论过，通过配置名就能理解它们的作用。

email

e-mail 插件是一个很重要的输出插件，可用于将特定的事件和失败的场景通过 e-mail 的形式发送。

基本的必选配置项如下：

```
email {
    to => "abc@example.com"
}
```

配置选项

email 输出插件有下面这些配置选项可用。

attachments

attachments参数指定了一组需要作为e-mail附件发送的文件路径。默认值是[]。

body

指定文本格式的邮件正文。默认值是""。

cc

cc 选项指定一组 e-mail 地址作为发送邮件的 cc 地址。可以用逗号分隔符来指定多个 e-mail 地址。

from

指定发送者的 e-mail 地址。默认值是"logstash.alert@nowhere.com"，（错误时）应该根据告警或系统的类型重写该参数。

to

to 选项是必选参数，指定邮件接收者的 e-mail 地址。可以用逗号分隔符来指定多个 e-mail 地址。

htmlbody

以 HTML 格式指定邮件的正文。可以在 e-mail 正文中包含 HTML 标签。

replyto

replyto 属性指定 e-mail 的 Reply-To 字段中需要的 e-mail 地址。

subject

subject 属性指定 e-mail 的主题。默认值是""。

elasticsearch

elasticsearch 插件是 ELK 技术栈中用到的最重要的插件，它用于将输出结果存储到 elasticsearch 搜索引擎中供 Kibana 分析。

我们会在*第五章 为什么需要 ELK 中的* Elasticsearch 中详细探讨 Elasticsearch，这里我

们先看一下 elasticsearch 插件的配置选项。

elasticsearch 输出插件的基础配置如下：

```
elasticsearch{
}
```

配置选项

一些最重要的配置选项如下表所列：

选项	数据类型	是否必选	默认值
action	字符串	否	“index”
bind_host	字符串	否	
bind_port	数字	否	
cacert	合法系统路径	否	
cluster	字符串	否	
document_id	字符串	否	
document_type	字符串	否	
host	数组	否	
index	字符串	否	“logstash-%{+YYYY.MM.dd}”
max_retries	数字	否	3
node_name	字符串	否	
password	密码	否	
port	字符串	否	
user	字符串	否	

ganglia

Ganglia 是一个用于监控分布式计算环境中集群性能的工具。Ganglia 会在每一台需要监控的机器上安装一个称为 Gmond 的后台服务进程。

Logstash 中，ganglia 输出插件会基于事件日志将监控指标发送给 gmond 服务进程。

ganglia 输出插件的基本配置如下：

```
ganglia {
    metric =>
    value =>
```

```
}
```

配置选项

ganglia 插件有下面这些配置选项可用。

metric

metric 选项指定了用于性能监控的指标。可以取事件的字段值作为这个选项的配置值。

unit

unit 选项指定了监控指标的单位，如 kb/s、ms。

value

value 选项指定了 metric 指标的值。

jira

Logstash 默认安装中并没有包括 jira 输出插件，但是可以用下面的命令安装：

```
bin/plugin install logstash-output-jira
```

jira 插件用于将事件发送给基于日志中特定事件创建 JIRA 标签的 JIRA 实例。jira 插件内部会使用 JIRA REST API 将输出事件发送给 JIRA 实例，因此 JIRA 实例需要配置成接受 REST API 调用。

jira 输出插件的基础配置如下：

```
jira {
    issuetypeid =>
    password =>
    priority =>
    projected =>
    summary =>
    username =>
}
```

配置选项

下面是 jira 插件的配置选项和数据类型：

选项	数据类型	是否必选
assignee	字符串	否
issuetypeid	字符串	是
password	字符串	是
priority	字符串	是
projectid	字符串	是
reporter	字符串	否
summary	字符串	是
username	字符串	是

kafka

正如在 Hortonworks 的 Kafka 主页(http://hortonworks.com/hadoop/kafka/)介绍的那样：

“Apache Kafka™ 是一个基于发布-订阅模式的快速、可扩展、持久化、支持容错的消息系统。”

kafka 输出插件用于将特定的事件写入到 kafka 的某个 topic 中。它使用了 Kafka Producer API 来将消息写入 topic。

kafka 插件的基本配置如下：

```
kafka {
    topic_id =>
}
```

配置选项

kafka 插件有许多配置选项，可以从官方文档中获取，topic_id 是这个插件唯一的必选参数。

topic_id

topic_id 选项指定了消息发送的目标 topic。

lumberjack

lumberjack 输出插件用于将事件输出到 Logstash 转发器或 lumberjack。

lumberjack 插件的基本配置如下：

```
lumberjack {
    hosts =>
    port =>
    ssl_certificat =>
}
```

配置选项

lumberjack 插件有下面这些配置选项可用：

hosts

hosts 选项以数组的形式指定一组 lumberjack 可以发送消息的主机地址。

port

port 选项指定连接 lumberjack 进行通信的端口。

ssl_certificate

指定用于通信的 SSL 证书的路径。

redis

redis 输出插件用于将事件输出到 redis 实例。

配置选项

redis 输出插件的配置选项和 redis 输入插件的配置选项类似。

rabbitmq



RabbitMQ 是一个开源的消息代理软件（有时也称为面向消息的中间件），它实现了高级消息队列协议（AMQP）。更多的信息可以从官方文档获取 http://www.rabbitmq.com。

在 RabbitMQ 中，生产者总是将消息发送给交换器（exchange），由交换器决定如何处理消息。有几种不同的交换器类型，对消息进行后续处理，它们分别是 direct、topic、headers 和 fanout。

rabbitmq 插件将事件从日志推送给 RabbitMQ 交换器。

rabbitmq 插件的基本配置如下：

```
rabbitmq {
    exchange =>
    exchange_type =>
    host =>
}
```

stdout

stdout 输出插件将事件输出到控制台。它经常用来调试 Logstash 配置，在将 Logstash 输出和其他系统进行整合之前进行测试。

基本配置如下：

```
output {
  stdout { }
}
```

mongodb

MongoDB 是一种面向文档的 NoSQL 数据库，以 JSON 文档的形式存储数据。

和 jira 插件一样，mongodb 插件也是一个由社区维护的插件，Logstash 默认安装没有包含这个插件。可以使用下面的指令安装插件：

```
bin/plugin install logstash-output-mongodb
```

mongodb 插件的基础配置如下：

```
mongodb {
    collection =>
    database =>
    uri =>
}
```

配置选项

mongodb 插件有下面这些配置选项可用。

collection

collection 选项指定将数据写出到 mongodb 中的哪个 collection。

database

database 选项指定了用于存储数据的 mongodb 数据库。

uri

uri 选项指定连接 mongodb 实例的连接串。

关于所有可用的 Logstash 输出插件的最新文档可以从这里获取：https://www.elastic.co/guide/en/logstash/current/output-plugins.html。

过滤插件

过滤插件用于在输出插件输出结果之前，对从输入插件中读取的事件进行中间处理。它们常用于识别输入事件的字段，并对输入事件的部分内容进行条件判断处理。

让我们看一些最重要的过滤插件。

csv

csv 过滤器用于将 CSV 文件输入的数据进行解析，并将值赋给字段。

配置选项

csv 过滤插件的配置选项已经在第 2 章 *构建第一个 ELK 数据管道*中介绍过。

date

ELK 中，给事件赋予正确的时间戳非常重要，只有这样才能在 Kibana 中使用时间过滤器对事件进行分析。date 过滤器基于日志中的字段将合适的时间戳赋给事件，并正确处理时间戳的格式。

如果没有使用 date 过滤插件，Logstash 会在第一次遇到事件或从文件中读取事件时将事件的时间设置为当前时间戳。

date 过滤插件的基本配置如下：

```
date {
}
```

配置选项

我们已经在第 2 章 构建第一个 ELK 数据管道中讲解过 date 过滤器的配置选项。

drop

drop 过滤器会将满足条件的所有事件都丢弃掉。

下面是一个例子：

```
filter {
if [ fieldname == "test"] {
drop {
}
}
}
```

这个过滤器会将 fieldname 值为 test 的所有事件都丢弃掉。这个插件可过滤输入事件中的无用信息。

配置选项

这个过滤插件有下面这些配置选项：

- add_field
- add_tag

- remove_field
- remove_tag

geoip

geoip 过滤器可基于输入事件中的 IP 地址给事件增加地理位置信息。这些信息从 Maxmind 数据库中获取。



Maxmind 是一个专门提供 IP 地址信息产品的公司。GeoIP 是它们开发的智能 IP 产品，用于 IP 地址的位置跟踪。所有 Logstash 版本都自带一个 Maxmind 的 GeoLite 城市数据库。这个地址数据库也可以从 http://dev.maxmind.com/geoip/legacy/geolite/获取。

geoip 过滤器的基础配置如下：

```
geoip {
    source =>
}
```

配置选项

geoip 插件有下面这些配置选项可用。

source

source 是一个字符串类型的必选选项，指定了需要使用 geoip 服务来进行映射的 IP 地址或主机名。可以指定使用事件中任何包含 IP 地址或主机名的字段，如果字段是数组类型，只会取数组中的第一个值进行 IP 地址映射。

grok

grok 插件是目前为止 Logstash 所拥有的最流行、最强大的插件。使用它可以解析任何非结构化的日志事件，并将日志转化成一系列结构化的字段，用于后续的日志处理和分析。

它可以用于解析任何类型的日志，包括 Apache 日志、MySQL 日志、自定义应用日志或者任何事件中的非结构化的文本。

Logstash 默认包含了很多 grok 模式，可以直接用来识别特定类型的字段，也支持自定义正则表达式。

所有可用的 grok 模式可以从这里获取：https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns。

下面是 grok 模式的一些例子：

```
HOSTNAME \b(?:[0-9A-Za-z][0-9A-Za-z-]{0,62})(?:\.(?:[0-9A-Za-z]
[0-9A-Za-z-]{0,62}))*(\.?|\b)
DAY (?:Mon(?:day)?|Tue(?:sday)?|Wed(?:nesday)?|Thu(?:rsday)?|Fri(?:day
)?|Sat(?:urday)?|Sun(?:day)?)
YEAR (?>\d\d){1,2}
HOUR (?:2[0123]|[01]?[0-9])
MINUTE (?:[0-5][0-9])
```

上面的 grok 模式可以使用下面这样的操作符来直接识别这些类型的字段：

```
%{HOSTNAME:host_name}
```

上面这个例子中，我们希望将日志事件中代表主机名的文本赋值给 host_name 这个字段。

接下来我们更详细地看一下 grok。

日志中的 grok 模式用这个通用的格式表示：

```
%{SYNTAX:SEMANTIC}
```

这里 SYNTAX 是用于匹配日志文件中文本的模式名字，SEMANTIC 是我们给匹配到的模式取的字段名。

我们以下面的例子为例。

假设你想表示事件传输的字节数：

```
%{NUMBER:bytes_transferred}
```

这里 bytes_transferred 字段指的是日志事件传输字节数的实际数值。

下面我们看一下如何用 grok 模式来表示一行 HTTP 日志：

```
54.3.245.1 GET /index.html 14562 0.056
```

grok 模式是这样的：

```
%{IP:client_ip} %{WORD: request_method} %{UNIPATHPARAM:uri_path}
%{NUMBER:bytes_transferred} %{NUMBER:duration}
```

匹配前面这个事件的 grok 配置如下：

```
filter {
grok {
match => { "message"=>"%{IP:client_ip} %{WORD: request_
method} %{UNIPATHPARAM:uri_path} %{NUMBER:bytes_transferred}
%{NUMBER:duration"}
}
}
}
```

使用 grok 过滤器处理上面的事件后，我们可以看到事件中增加了如下字段和值：

- client_ip: 54.3.245.1
- request_method: GET
- uri_path: /index.html
- bytes_transferred: 14562
- duration: 0.056

自定义 grok 模式

如果 grok 模式中没有需要的模式，可以使用正则表达式创建自定义模式。

下面这些 URL 对设计和测试 grok 模式、匹配文本非常有用：http://grokdebug.herokuapp.com 和 http://grokconstructor.appspot.com。

mutate

mutate 是一个重要的过滤插件，可用于对输入事件进行重命名、移除、替换和修改字段。它也用于转换字段的数据类型、合并两个字段、将文本从小写转换为大写或者从大

写转换为小写。

mutate 过滤器的基本配置如下：

```
filter {
mutate {
}
}
```

配置选项

mutate 过滤器有很多配置选项，大多数可以通过配置名理解其作用：

选项	数据类型	是否必选	默认值
add_field	哈希	否	{ }
add_tag	数组	否	[]
convert	哈希	否	
join	哈希	否	
lowercase	数组	否	
merge	哈希	否	
remove_field	数组	否	[]
remove_tag	数组	否	[]
rename	哈希	否	
replace	哈希	否	
split	哈希	否	
strip	数组	否	
update	哈希	否	
uppercase	数组	否	

sleep

sleep 过滤插件用于将 Logstash 置于 sleep 模式，sleep 的时间由参数指定。另外也可以基于事件数指定 sleep 的频率。

下面是一个例子。

如果我们希望 Logstash 每处理五个事件就 sleep 一秒，我们可以这样配置过滤器：

```
filter {
  sleep {
    time => "1" #sleep 一秒
    every => 5 #每处理五个事件就sleep一次
  }
}
```

所有关于可用的 Logstash 过滤插件的列表和最新的文档可以从这里获取 https://www.elastic.co/guide/en/logstash/current/filter-plugins.html。

编解码插件

编解码插件用于对输入事件进行解码，对输出事件进行解码。它们以流式过滤器的形式在输入插件和输出插件中工作。

重要的编解码插件包括：

- avro
- json
- line
- multiline
- plain
- rubydebug
- spool

我们来探讨一些最常用的编解码插件的细节。

json

如果输入事件或输出事件是完整的 JSON 文档，json 编解码插件非常有用。它可以这样配置：

```
input {
stdin {
codec => json {
}
}
}
```

也可以这样配置：

```
input {
stdin {
codec => "json"
}
}
```

line

line 编解码器用于将每行输入日志作为一个事件，将每个输出事件解码成一行。它可以这样配置：

```
input {
stdin {
codec => line {
}
}
}
```

也可以这样配置：

```
input {
stdin {
codec => "line"
}
}
```

multiline

如果需要把多行日志作为一个事件处理，multiline 编解码插件会非常有用。这在处理 Java 异常或者调用栈跟踪时非常方便。

例如，下面的配置会将完整的调用栈跟踪作为一个事件：

```
input {
  file {
    path => "/var/log/someapp.log"

    codec => multiline {
```

```
        pattern => "^%{TIMESTAMP_ISO8601}"
        negate => true
        what => previous
      }
    }
  }
```

上面的配置会将所有不以时间戳开头的日志行作为上一行日志的一部分，整个记录为一个单独的事件。

plain

plain 编解码插件用于指示输入或输出事件时，不需要额外的编解码操作，对应的输入输出插件会自己处理编解码。对很多插件如 redis、mongodb 等而言，plain 是默认的编解码类型。

rubydebug

rubydebug 插件只在输出事件时使用，它使用 Ruby Awesome 打印库打印输出事件。

关于所有编解码插件的完整列表和最新的文档可以从这个地址获取：https://www.elastic.co/guide/en/logstash/current/codec-plugins.html。

总结

这一章我们学习各种 Logstash 插件的配置，包括输入插件、过滤插件、输出插件和编解码插件，了解了如何使用这些插件对不同来源的多种类型的日志进行采集、解析和转换。

下一章，我们将学习如何创建自定义插件，以满足处理自定义格式的需求，或者处理现有插件无法处理的特殊事件类型。

4

创建自定义 Logstash 插件

在前一章，我们已经看到了如何使用 Logstash 提供的丰富插件来解决不同类型日志数据的输入、处理和输出需求。如果有个性化的数据处理需求，可以通过自定义插件来满足。在这一章，我们会考虑下面这些 Logstash 插件的高级概念：

- Logstash 插件管理。
- 下载和安装社区维护的插件。
- 创建自定义 Logstash 插件。

Logstash 插件管理

从 1.5.0+版本开始，Logstash 插件和核心模块分开维护，每个插件都使用自包含的 **RubyGems** 包来维护。这便于将插件的版本发布和 Logstash 版本发布独立开来，也减少了 Logstash 核心包的总体大小。

Logstash 插件使用 ruby 开发。



RubyGems 是 Ruby 编程语言的包管理器，它提供了一种发布 Ruby 程序和库的标准格式（以一种自包含的称为“gem”的格式）。它被设计成一种用于方便管理包的安装和分发包的工具。

Logstash 核心插件和社区插件发布在 https://rubygems.org/，可以从这里方便的下载和

安装插件。

所有 Logstash 插件存储在下面这个 GitHub 的仓库中：https://github.com/logstash-plugins。

插件生命周期管理

Logstash 插件管理通过 Logstash 安装包提供的安装脚本实现：

```
$LOGSTASH_HOME/bin/plugin
```

安装插件

我们可以使用下面这个命令安装插件：

```
$bin/plugin install <plug_in_name>
```

`plug_in_name` 是在 https://rubygems.org/或 Logstash 插件仓库中 gem 文件里命名的插件名。

我们以下面的命令作为案例：

```
$bin/plugin install logstash-input-rabbitmq
```

上面的命令会安装 `rabbitmq` 输入插件。你还可以通过`--version` 参数指定安装一个特定版本的插件。

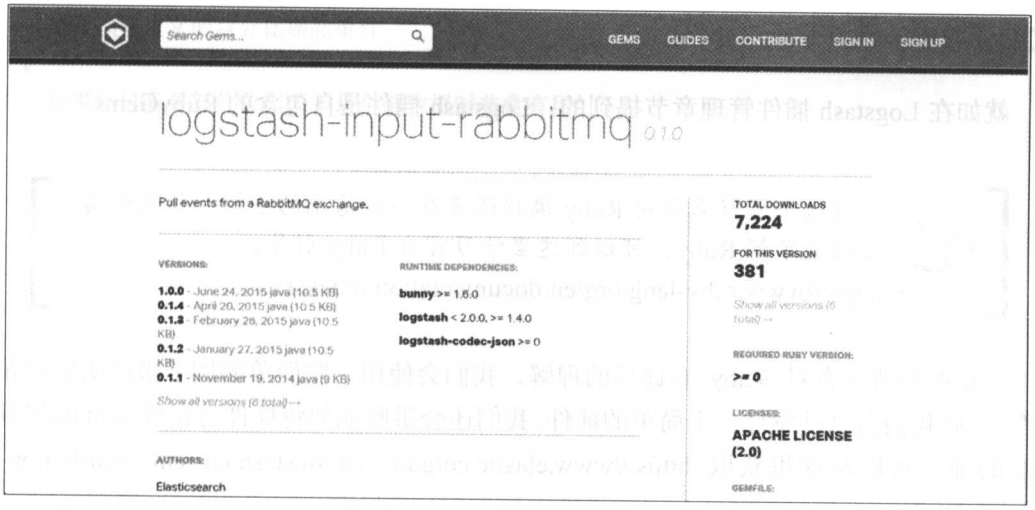


RabbitMq(https://www.rabbitmq.com)是一个消息中间件，提供发送消息、接收消息的通用平台，消息会一直存在系统中，直到被接收。

从 https://rubygems.org/下载的插件也可以像下面这样通过文件路径的方式安装：

```
$bin/plugin install path/to/logstash-input-rabbitmq-0.1.0.gem
```

你还可以在 https://rubygems.org/搜索关键字"`logstash`"，浏览所有 Logstash 插件。



插件下载页面 https://rubygems.org

更新插件

我们可以使用下面的命令更新一个已经安装好的插件：

```
$bin/plugin update <plug_in_name>
```

我们以下面的命令为例：

```
$bin/plugin update logstash-input-rabbitmq
```

上面的命令会将 `rabbitmq` 插件更新到最新版本。请确保在将更新安装到生产环境之前进行测试。

卸载插件

我们可以使用下面这个命令卸载插件：

```
$bin/plugin uninstall <plug_in_name>
```

以下面的命令为例：

```
$bin/plugin uninstall logstash-input-rabbitmq
```

上面的命令会卸载 `logstash-input-rabbitmq` 插件。

Logstash 插件的结构

就如在 Logstash 插件管理章节提到的，Logstash 插件是自包含的 RubyGems。



阅读这个章节需要对 Ruby 编程语言有一些基础的了解。如果你需要快速了解 Ruby，可以到这里学习官方 Ruby 引导：https://www.ruby-lang.org/en/documentation/quickstart/。

本章不期待读者对 Ruby 有详尽的理解，我们会使用一些简单的例子来说明插件如何工作，然后我们会自己设计一个简单的插件。我们还会讲解如何将插件打包成 gem 的细节。更多的细节可以从这里获取 https://www.elastic.co/guide/en/logstash/current/contribution-to-logstash.html。

我们看一下 drop 过滤器插件的结构，drop 插件用于丢弃满足特定条件的事件（https://github.com/logstash-plugins/logstash-filter-drop）：

```
# encoding: utf-8
require "logstash/filters/base"
require "logstash/namespace"

# 删除事件过滤器.
#
# 删除所有传递到该过滤器的事件.
#
# 最好和条件语句一起使用，例如：
# [source,ruby]
#     filter {
#       if [loglevel] == "debug" {
#         drop { }
#       }
#     }
#
# 以上配置只会将loglevel字段值是
# `debug`的时间传给删除过滤器。这会导致所有符合条件的事件都被删除
  class LogStash::Filters::Drop < LogStash::Filters::Base
  config_name "drop"
```

```
# 根据预先定义的百分比删除事件
#
# 如果你只需要部分数据而不是全部数据时非常有用
#
# 比如，删除40%的loglevel字段值是"debug"的事件。
#
#     filter {
#       if [loglevel] == "debug" {
#         drop {
#           percentage => 40
#         }
#       }
#     }
config :percentage, :validate => :number, :default => 100
public
def register
  # 什么也不做
end

public
def filter(event)
  event.cancel if (@percentage == 100 || rand < (@percentage /
100.0))
  end # def filter
end # class LogStash::Filters::Drop
```

接下来我们来分析插件的每一部分。

需要的依赖

第一个 require 语句加载了文件 logstash/namespace.rb，这个文件定义了输入、过滤器、输出和编解码插件的模块命名空间。

```
require "logstash/namespace"
```

然后我们加载了过滤器插件的依赖：

```
require "logstash/filters/base"
```

对于输入插件，需要加载"/logstash/inputs/base"，输出插件需要加载"/logstashoutputs/base"。

类定义

我们需要给每一个插件定义一个类，对于过滤器插件，需要以 Base 类作为基类：

```
class Logstash::Filters::Drop < Logstash ::Filters::Base
```

我们实现的是 drop 过滤器，就以 Drop 作为这个类的名字。

配置插件名字

接下来需要指定我们会在 Logstash 配置中使用的插件名字，我们通过 config_name 来定义插件的名字：

```
config_name "drop"
```

配置文件中我们可以这样使用这个插件：

```
filter {
drop {
}
}
```

配置选项设置

我们可以按需求定义任意多的配置选项。可以设置配置选项的名字、数据类型、默认值，以及是否必选选项：

```
config :percentage, :validate => :number, :default => 100
```

下面是配置选项。

- :validate：这个配置指定参数的数据类型，可以取:string, :number, :array, :hash, :boolean 等。对于 drop 过滤器，配置项 percentage 的数据类型是数字类型。
- :default：指定配置项的默认值。对于 drop 过滤器，配置项 percentage 的默认值是 100。

- :required：这是一个布尔型参数，取 true 或 false，用于指定配置项是否必选。

插件方法

每一种类型的插件（输入、过滤器、输出、编解码）都需要实现一些方法，用于初始化实例变量、执行插件中的真正的操作。

插件类型	方法
输入插件	register 和 run（queue）
过滤器插件	register 和 filter（event）
输出插件	register 和 receive
编解码插件	register, encode,decode

输入插件

对于输入插件，我们要实现 register 和 run（queue）这两个方法。

register 方法用于初始化所需要的实例变量。

run 方法将输入消息流转换成事件，供后续处理：

```
public
def run(queue)
  # 将输入消息转换为事件的代码
end #def run
```

过滤插件

对于过滤器插件，我们需要实现 register 和 filter（event）这两个方法：

```
public
def register
  # 什么也不做
end
```

register 方法用于初始化所需要的实例变量。对于 drop 过滤器插件，我们没有用到任何实例变量，所以这个方法是空的。

```
public
def filter(event)
      event.cancel if (@percentage ==100 || rand < (@percentage/100.0))
end #def filter
```

filter 方法用于执行过滤事件的真正操作。在 filter 方法里，我们可以使用通过 config 设置的参数，参数名前面需要加上@这个前缀，事件的属性可以通过事件哈希表访问。

比如我们可以用 event["message"]这样的代码来访问事件的 message 属性。

event.cancel 这类操作也可以使用。

在 drop 过滤器中，我们使用 event.cancel 来丢弃事件。

输出插件

对于输出插件，我们需要实现 register 和 receive 这两个方法。

register 方法用于初始化所需要的实例变量。

receive 方法取决于插件的类型，会在事件输出之前调用。

```
public
def receive(event)
end #def event
```

编解码插件

编解码插件和输入插件、输出插件一起工作，对输入事件进行解码操作，对输出事件进行编码操作。

对于编解码插件，我们需要实现 register、encode 和 decode 这几个方法。

register 方法用于初始化所需要的实例变量。

encode 方法用于将事件编码成某种格式。

如 json 编解码插件会将事件转换成 json 格式：

```
public
  def encode(event)
    @on_event.call(event, event.to_json)
  end
```

decode 方法会对输入数据进行解码。这个方法需要使用 yield 语句将解码后的事件输送到管道。

例如 spool 编解码插件中，我们使用 yield 将消息输送到 buffer 中：

```
public
  def decode(data)
  data.each do |event|
    yield event
  end
end
```

实现一个 Logstash 过滤器插件

现在我们已经了解了 Logstash 插件的结构，接下来可以开发一个我们自己的插件。

在这一节里面，我们会使用前面学到的插件结构的知识，实现一个简单的过滤器插件。

在这个示例中，假设输入流里面是一系列数字，我们将这些数字作为某种货币的金额，货币名称通过插件的配置参数设置。下面是这个 currency 过滤器插件的代码：

```
# 在价格字段中增加货币符号
#
#filter {
#     currency{
#          name => "$"
#     }
#}

require "logstash/filters/base"
require "logstash/namespace"

class LogStash::Filters::Currency < LogStash::Filters::Base
```

```
config_name "currency"

config :name, :validate => :string, :default => "$"

public
def register
#do nothing
end

public
def filter(event)
    if @name
        msg = @name + event["message"]
        event["message"] = msg
    end
end
end
```

我们来观察前面的过滤器结构。

首先我们加载插件需要的依赖包：

```
require "logstash/filters/base"
require "logstash/namespace"
```

接着为这个过滤器定义一个类：

```
class LogStash::Filters::Currency < Logstash::Filters::Base
```

接下来使用 config_name 配置过滤器的名字：

```
config_name "currency"
```

接下来指定这个插件的配置选项，我们需要配置货币的名称，将其增加到消息中。我们这样定义：

```
config :name, :validate => :string, :default => "$"
```

我们没有用到任何实例变量，所以 register 方法是空的：

```
public
```

```
def register
# 什么也不做
end
```

接下来实现过滤器插件的 filter 方法，给事件增加货币符号：

```
public
def filter(event)
    if @name
        msg = @name + event["message"]
        event["message"] = msg
    end
end
```

在 filter 方法里，我们先检查插件的 name 配置选项是否有值。如果有值，就把配置的值加到消息前面，否则不做任何处理。

过滤器可以像下面这样使用：

```
filter {
    currency {
        name => "$"
    }
}
```

假设输入的是 200，经过这个过滤器处理后，输出事件会变成这样：

```
{
   "@timestamp"=> "2015-06-21T14:21:54.123Z",
   "message"=> "$200",
}
```

构建插件

现在我们已经创建了一个插件，保存在文件 currency.rb 中，目录结构如下：

```
logstash-filter-currency
└──lib
|   └──logstash
|       └──filters
```

```
|  └──currency.rb
Gemfile
logstash-filter-currency.gemspec
```

在给插件创建 RubyGem 前，我们需要在 `logstash-filter-currency` 文件夹中创建 gemfile 和 gemspec 文件。



gemfile：gemfile 用于描述执行相关 Ruby 代码需要的 gem 依赖关系。

Gemspec 文件：gemspec 文件定义 RubyGem 的规格。

我们在 gemspec 文件中增加相关规格：

```
Gem::Specification.new do |s|
  s.name = 'logstash-filter-currency'
  s.version        = '0.1.0'
  s.licenses = ['Apache License (2.0)']
  s.summary = "This plugin adds a currency name before message."
  s.description = "This plugin is used to add core logstash available
plugin, to define a new functionality of adding currency
symbols for certain messages"
  s.authors = ["Saurabh Chhajed"]
  s.email = 'saurabh.chhajed@gmail.com'
  s.homepage = "http://saurzcode.in"
  s.require_paths = ["lib"]

  # Files
  s.files = ["lib/logstash/filters/currency.rb"]

  # Special flag to let us know this is actually a logstash plugin
  s.metadata = { "logstash_plugin" => "true", "logstash_group" =>
"filter" }

  # Gem dependencies
  s.add_runtime_dependency "logstash-core", '>= 1.4.0', '< 2.0.0'
  s.add_development_dependency 'logstash-devutils'
```

```
end
```

将上面的内容保存到 logstash-filter-currency.gemspec 文件。

需要使用 Ruby gem 打包器构建 gem 文件，打包器可以使用下面的命令安装：

```
$ gem install bundler
```

关于打包器的更多信息可以从这里获取：http://bundler.io/。

现在可以使用下面的命令构建 gem 文件：

```
$gem build logstash-filter-currency.gemspec
```

上面的命令会在 logstash-filter-currency 目录中创建一个 logstash-filter-currency-0.1.0.gem 文件。

可以使用下面的命令安装这个 gem：

```
$bin/plugin install /path/to/logstash-filter-currency-0.1.0.gem
```

如果安装成功，可以通过下面的命令查看这个插件：

```
$bin/plugin list
```

我们可以使用 logstash -e 选项迅速测试这个插件：

```
bin/logstash -e 'input{stdin{}} filter { currency {name => "%"}} output {stdout
{codec => rubydebug}}'
```

这个过滤器插件会在任何一个我们输入的数字前加上$这个货币符号：

```
200
{
        "message" => "$200"
        "@version" => "1",
        "@timestamp" => "2015-06-27T19:17:20.230Z",
        "host" => "saurzcode"
}
```

可以看到，在控制台输入 200 这个数字后，$符号加到了 200 之前。

现在我们已经成功地创建了一个 Logstash 过滤器插件，并进行了测试。

可以通过类似的方式创建和部署输入插件和输出插件。

总结

在这一章里，我们学习了如何创建自定义 Logstash 插件，实现现有插件无法满足的需求。我们了解了所有关于 Logstash 支持的不同类型的插件和特性，还知道了如何扩展 Logstash，以满足不同的输入输出源的多种需求。

接下来，我们会详细讲解 ELK 套件中另外两个组件——Elasticsearch 和 Kibana 的特性。

5

为什么需要 ELK 中的 Elasticsearch

在这一章中，我们来看一下 Elasticsearch 在 ELK 技术栈中的作用。本章涵盖了 Elasticsearch 的特点，以及为什么这个出色的技术能够快速响应实时分析的搜索需求。最后，还将简要地介绍一些 Elasticsearch 可用的插件，这些插件使得我们在处理 Elasticsearch 集群时更容易。

为什么是 Elasticsearch

Elasticsearch 是一种在分布式环境中快速、可扩展的搜索和分析引擎。我们已经在第 1 章 EKL 技术栈介绍中提到，一些很大的分布式架构的公司，如 GitHub、StackOverflow、维基百科，都在使用 Elasticsearch 的全文搜索、结构化搜索和快速相关性搜索的分析能力。

Elasticsearch 建立在 Apache Lucene 之上。Lucene 的定义在 Apache Lucene 网页（https://lucene.apache.org）上这样写道：

> “Apache lucene™ 是一种高性能、全功能的完全用 java 写的文本搜索引擎库。它是一种几乎适合于任何需要全文搜索，特别是跨平台的应用程序的技术”

Elasticsearch 通过提供强大的 RESTful API 隐藏了 Lucene 背后的复杂性，使得查询索引数据更容易，并使其适用于任何编程语言。Elasticsearch 可以基于 PB 级别的结构化或者非结构化数据建立跨多台服务器的分布式实时分析，这大大扩展了 Lucene 的能力。

在深入了解 Elasticsearch 提供的各种 API 之前，让我们先了解一些 Elasticsearch 的基本概念。

Elasticsearch 的基本概念

让我们来看一些 Elasticsearch 的基本概念，它们会阐释 Elasticsearch 是如何存储索引数据的。

索引

在 Elasticsearch 中，索引是具备某些共同特征的文档集。

每一个索引包含多个类型，每个类型相应的包含多个文档，每个文档又包含多个字段。在 Elasticsearch 中，一个索引包含多个 JSON 格式的文档。在 Elasticsearch 集群中的索引数量可以是任意的。

在 ELK 中，将 Logstash 的 JSON 文档发送到 Elasticsearch 时，它们被存储为默认的索引模式"logstash-% {+YYYY.MM.dd }"。索引按天分区，可以根据需要简单地搜索和删除。这个索引模式可以在 Logstash 的输出插件的配置中修改。

搜索和查询索引的 URL 看起来如下：

```
http://localhost:9200/[索引]/[类型]/[操作]
```

文档

Elasticsearch 中的文档是指存储在索引里的 JSON 格式的文档。每一个文档都有一个类型和相应的唯一标识（ID）。

例如，存储在 Elasticsearch 的文档看起来如下：

```
{
  "_index" : "packtpub",
  "_type" : "elk",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source":{
book_name : "learning elk"
}
}
```

字段

字段是文档内的基本单元。正如在前面的例子中，一个基本字段是如下的键值对：

```
book_name : "learning elk"
```

类型

类型用于提供索引中的逻辑分区。它基本上代表一类类似的文档类型。一个索引可以有多个类型，我们可以根据上下文来定义它们。

例如，Facebook 的索引中可以使用 `post` 作为一种索引类型，使用 `comments` 作为另一种索引类型。

映射

映射是用来映射文档的每个字段，以及字段对应的数据类型，如 `string`、`integer`、`float`、`double`、`date` 等等。Elasticsearch 会在创建索引时自动创建字段的映射，这些映射可以根据某些特定类型的需求方便地查询或修改。

分片

分片是存储索引的实际物理实体。每个索引可以有多个存储数据的主分片和副本分片。分片分布在集群的所有节点之间，并且在有节点失效或新节点加入时，可以从一个节点移动到另一节点。

主分片和副本分片

在一个 Elasticsearch 索引中每个文件存储在一个主碎片和一些副本分片中。而索引文件先存储在主分片中，然后再存储到相应的副本分片中。默认情况下，每个索引的主分片数量是 5，当然我们也可以根据需要自行配置。

副本分片通常与主分片驻留在不同的节点上，以便于满足多个请求情况下的故障转移和负载平衡。

集群

集群是存储索引数据的节点集合。Elasticsearch 通过将数据存储在集群中来提供水平扩展能力。每个集群都有一个集群名称来表示，以便于不同的节点辨识连接。集群名称可以通过 `elasticsearch.yml` 配置文件中名为 `cluster.name` 的属性配置，默认为 `"Elasticsearch"`：

```
cluster.name: elasticsearch
```

节点

节点是一个单一的 Elasticsearch 运行实例，它属于某个集群。默认情况下，每个节点都会加入名为`"elasticsearch"`的集群。在 `elasticsearch.yml` 中，每个节点有自己的配置，包括内存和资源分配的不同设置。

在 Elasticsearch 中，节点可以扮演如下三种角色。

- **数据节点**：数据节点用于索引文档，以及对这些文档执行搜索操作。如果要提升性能或者扩展集群，通常建议添加更多的数据节点。在 elasticsearch.yml 配置文件中设置如下的节点属性，就可以将节点变成数据节点：

```
node.master = false
node.data=true
```

- **主节点**：主节点负责集群的管理。对于大集群，建议有三个专用的主节点（一主两备），它们只作为主节点，不存储索引或执行搜索。在 `elasticsearch.yml` 配置文件中做如下设置，就可以将节点变成专用的主节点：

```
node.master =true
node.data=false
```

- **路由节点或负载平衡节点**：这些节点既不作为主节点也不担当数据节点角色，只用于负载均衡，或路由搜索请求，又或将文档索引到适当的节点。这对于高负荷的搜索或索引操作非常有用。在 `elasticsearch.yml` 配置文件中做如下设置，就可以将节点变成路由节点：

```
node.master =false
node.data=false
```

探索 Elasticsearch API

在 ELK 中，虽然 Logstash 和 Kinana 充当着与 Elasticsearch 索引进行交流的接口，但是我们依然有必要了解 Logstash 和 Kibana 是如何使用 Elasticsearch RESTful APIs 来执行多种操作的，比如创建和管理索引、存储和检索文档，以及围绕索引形成各种类型的搜索查询语句。另外，知道如何删除索引也是非常有用的。

我们知道 Elasticsearch 提供了大量的 API 来执行各种操作。用命令行查询集群的常见语法如下：

```
$curl -X<VERB>
'<PROTOCOL>://<HOST>:<PORT>/<PATH>/<OPERATION_NAME>?<QUERY_STRING>' -
d '<BODY>'
```

让我们来了解这个命令的各个部分。

- VERB：请求方法的类型，包括 GET、POST、PUT、DELETE、HEAD。
- PROTOCOL：协议类型，要么是 http，要么是 https。
- HOST：节点在集群中的主机名，对于本地安装可以是'localhost'或者'127.0.0.1'。
- PORT：Elasticsearch 实例运行的端口，默认是 9200。
- PATH：索引名、类型和要查询的文档 ID，例如：/index/type/id。
- OPERATION_NAME：要执行的操作的名称，例如：_search，_count 等等。
- QUERY_STRING：这是一个在查询参数中指定的可选参数。例如，?pretty 用于完美输出 JSON 文档。
- BODY：用于请求正文文本。

以下面的命令为例：

```
curl -XGET 'http://localhost:9200/logstash-2014.08.04/_search?pretty'
```

该 URL 将会搜索名为 logstash-2014.08.04 的索引。

接下来的章节中，假定已经按照 第 1 章 ELK 技术栈介绍 里提到的方法安装好 Elasticsearch，并且能正常运行。

在这一节里，将使用在 第 2 章 构建第一个 ELK 数据管道 里创建的示例索引，尝试

对其执行一些操作。

列出所有可用索引

首先尝试通过如下命令来查看集群中的所有可用索引：

```
curl -XGET 'localhost:9200/_cat/indices?v'
```

执行以上命令，我们将得到如下结果：

```
health status index           pri rep docs.count docs.deleted store.size
pri.store.size
green open   logstash-2014.12.19   5   1          1            0
6.1kb          6.1kb
green open   logstash-2014.12.08   5   1          1            0
6.1kb          6.1kb
green open   logstash-2014.07.17   5   1          1            0
6kb            6kb
green open   logstash-2014.08.04   5   1          1            0
6.1kb          6.1kb
green open   logstash-2014.11.05   5   1          1            0
6.1kb          6.1kb
green open   logstash-2014.07.27   5   1          1            0
6.1kb          6.1kb
green open   logstash-2014.09.16   5   1          1            0
6.1kb          6.1kb
green open   logstash-2014.12.15   5   1          1            0
6.1kb          6.1kb
green open   logstash-2014.12.10   5   1          1            0
6.1kb          6.1kb
green open   logstash-2014.09.18   5   1          1            0
6kb            6kb
green open   logstash-2014.12.18   5   1          1            0
6.1kb          6.1kb
green open   logstash-2014.07.08   5   1          1            0
6.1kb          6.1kb
```

这将显示存储在集群的节点中的所有索引，以及索引相关的信息，如健康值、索引名

称、大小、文档的数量、主分片的数量等等。

例如，上文中的第一行显示，名为 logstash-2014.12.19 的索引拥有 5 个主分片和 1 个副本分片，并且有 1 个正常文档和 0 个已删除文档。

列出集群中的所有节点

调用如下命令可以查看集群中的所有节点：

```
curl -XGET 'http://localhost:9200/_cat/nodes?v'
```

结果如下：

```
host       ip      heap.percent   ram.percent load node.role
master   name
packtpub   127.0.1.1   18             35       0.27     d
*        Animus
```

由于这是本地的一个单节点集群，所以这里显示了一个节点，以及该节点和内存相关的特征。

检查集群的健康状态

调用如下命令可以查看集群的健康状态：

```
curl -XGET 'http://localhost:9200/_cluster/health?pretty=true'
{
  "cluster_name" : "elasticsearch",
  "status" : "yellow",
  "timed_out" : false,
  "number_of_nodes" : 1,
  "number_of_data_nodes" : 1,
  "active_primary_shards" : 11,
  "active_shards" : 11,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 11
}
```

可以从集群层面、分片层面，或者索引层面来检查健康状态，使用类似如下的 URL 即可：

```
curl -XGET 'http://localhost:9200/_cluster/health?level=cluster&pretty=tr
ue'
curl -XGET 'http://localhost:9200/_cluster/health?level=shards&pretty=tr
ue'
curl -XGET 'http://localhost:9200/_cluster/health?level=indices&pretty=tr
ue'
```

集群的健康状态

Elasticsearch 集群健康状态由三个参数表示：

- **红色**表示部分或全部的主分片尚未准备好提供服务。
- **黄色**表明所有的主分片已成功分配，但部分或全部的副本分片尚未分配。一般情况下，单节点集群始终是黄色的健康状态，因为没有副本分片节点。
- **绿色**表明所有的主分片和副本分片都分配成功，并且集群正常运作。

创建索引

在 ELK 中，索引是根据在 Logstash 的 `elasticsearch` 输出插件中提供的索引名称自动创建的。尽管如此，还是来看一下如何手工创建索引：

```
curl -XPUT 'localhost:9200/<index_name>?pretty'
```

假如要创建一个名为 `packtpub` 的索引，可以执行以下命令：

```
curl -XPUT 'localhost:9200/packtpub/?pretty'
```

另外，还可以在往索引内写入文档的同时，直接创建索引，操作如下：

```
curl -xPUT 'localhost:9200/packtpub/elk/1?pretty' -d '
{
book_name : "learning elk"
}'
```

执行命令后的返回结果如下：

```
{
  "_index" : "packtpub",
  "_type" : "elk",
  "_id" : "1",
  "_version" : 1,
  "created" : true
}
```

通过上述命令，我们成功的创建了一个名为 packtpub 的索引，类型为 elk，并且存入了一个 ID 为 1 的文档。

检索文档

现在，可以来检索一下刚刚创建的索引：

```
curl -XGET 'localhost:9200/packtpub/elk/1?pretty'
```

执行以上命令后的返回结果如下：

```
{
  "_index" : "packtpub",
  "_type" : "elk",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source":{
 book_name : "learning elk"
 }
 }
```

_source 字段包含了完整的文档，文档在索引中的 ID 为 1。

让我们根据 *第 2 章 构建第一个 ELK 数据管道* 里的 GOOG 股票价格的索引来查询文档：

```
curl -XGET 'localhost:9200/logstash-2014.08.04/logs/_search?pretty'
```

将得到如下结果：

```
{
  "took" : 3,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 1.0,
    "hits" : [ {
      "_index" : "logstash-2014.08.04",
      "_type" : "logs",
      "_id" : "AU2qgZixPoayDyQnreXd",
      "_score" : 1.0,
      "_source":{"message":["2014-08-05,570.05255,571.9826,562.61255,
565.07257,1551200,565.07257"],"@version":"1","@timestamp":"2014-08-
04T23:00:00.000Z","host":"packtpub","path":"/opt/logstash/input/
GOOG.csv","date_of_record":"2014-08-05","open":570.05255,"high":5
71.9826,"low":562.61255,"close":565.07257,"volume":1551200,"adj_
close":"565.07257"}
} ]
}
}
```

可以看到_source字段里存储了完整的消息，包含从Logstash发出的JSON。

删除文档

执行如下命令可以删除索引内的文档：

```
curl -XDELETE 'localhost:9200/packtpub/elk/1?pretty'
```

执行命令后的返回如下：

```
{
   "acknowledged" : true
```

```
}
```

这个结果表明该索引已成功删除。

Elasticsearch Query DSL

到目前为止，所看到的用于检索数据的命令都是一些基本命令，但是 Elasticsearch 的询语句的强大之处在于，其是一个基于 JSON 的领域特定语言，称为 Query DSL。Kibana 为了得到特定格式的结果，广泛使用了 Query DSL。你几乎再也不必担心如何撰写查询语句的 JSON，因为 Kibana 会自动帮你创建，并且将结果格式化。

例如，如果只想要返回 3 条匹配结果，可以如下处理。

```
curl -XPOST 'localhost:9200/logstash-*/_search' -d '
{
  "query": { "match_all": {} },
  "size": 3
}'
```

以下返回也将只包含 3 条对应查询结果：

```
{
  "took": 390,
  "timed_out": false,
  "_shards": {
    "total": 640,
    "successful": 640,
    "failed": 0
  },
  "hits": {
    "total": 128,
    "max_score": 1,
    "hits": [
      {
        "_index": "logstash-2014.07.01",
        "_type": "logs",
        "_id": "AU2qge3cPoayDyQnreX0",
```

```
        "_score": 1,
        "_source": {
          "message":
["2014-07-02,583.3526,585.44269,580.39264,582.33765,1056400,582.33765"],
          "@version": "1",
          "@timestamp": "2014-07-01T23:00:00.000Z",
          "host": "packtpub",
          "path": "/opt/logstash/input/GOOG.csv",
          "date_of_record": "2014-07-02",
          "open": 583.3526,
          "high": 585.44269,
          "low": 580.39264,
          "close": 582.33765,
          "volume": 1056400,
          "adj_close": "582.33765"
        }
      },{
        "_index": "logstash-2014.07.09",
        "_type": "logs",
        "_id": "AU2qge3cPoayDyQnreXv",
        "_score": 1,
        "_source": {
          "message":
["2014-07-10,565.91254,576.59265,565.01257,571.10254,1356700,571.10254"],
          "@version": "1",
          "@timestamp": "2014-07-09T23:00:00.000Z",
          "host": "packtpub",
          "path": "/opt/logstash/input/GOOG.csv",
          "date_of_record": "2014-07-10",
          "open": 565.91254,
          "high": 576.59265,
          "low": 565.01257,
          "close": 571.10254,
          "volume": 1356700,
          "adj_close": "571.10254"
        }
      },{
```

```
          "_index": "logstash-2014.07.21",
          "_type": "logs",
          "_id": "AU2qgZixPoayDyQnreXn",
          "_score": 1,
          "_source": {
            "message":
["2014-07-22,590.72266,599.65271,590.60266,594.74268,1699200,594.74268"],
            "@version": "1",
            "@timestamp": "2014-07-21T23:00:00.000Z",
            "host": "packtpub",
            "path": "/opt/logstash/input/GOOG.csv",
            "date_of_record": "2014-07-22",
            "open": 590.72266,
            "high": 599.65271,
            "low": 590.60266,
            "close": 594.74268,
            "volume": 1699200,
            "adj_close": "594.74268"
          }
        }
      ]
    }
  }
```

同样的，想要结果按照某个字段进行排序的话，可以如下处理：

```
curl -XPOST 'localhost:9200/logstash-*/_search' -d '
{
"query" : {
"match_all" :{}
},
"sort" : {"open" : { "order":"desc"}},
"size" :3
}'
```

执行以上命令后，会看到返回的结果已经根据"open"字段降序排列：

```
{
  "took": 356,
  "timed_out": false,
  "_shards": {
    "total": 640,
    "successful": 640,
    "failed": 0
  },
  "hits": {
    "total": 128,
    "max_score": null,
    "hits": [
      {
        "_index": "logstash-2014.07.23",
        "_type": "logs",
        "_id": "AU2qgZixPoayDyQnreXl",
        "_score": null,
        "_source": {
          "message":
["2014-07-24,596.4527,599.50269,591.77271,593.35266,1035100,593.35266"],
          "@version": "1",
          "@timestamp": "2014-07-23T23:00:00.000Z",
          "host": "packtpub",
          "path": "/opt/logstash/input/GOOG.csv",
          "date_of_record": "2014-07-24",
          "open": 596.4527,
          "high": 599.50269,
          "low": 591.77271,
          "close": 593.35266,
          "volume": 1035100,
          "adj_close": "593.35266"
        },
        "sort": [
          596.4527
        ]
      },{
        "_index": "logstash-2014.09.21",
```

```
        "_type": "logs",
        "_id": "AU2qgZioPoayDyQnreW8",
        "_score": null,
        "_source": {
          "message":
["2014-09-22,593.82269,593.95166,583.46271,587.37262,1689500,587.37262"],
          "@version": "1",
          "@timestamp": "2014-09-21T23:00:00.000Z",
          "host": "packtpub",
          "path": "/opt/logstash/input/GOOG.csv",
          "date_of_record": "2014-09-22",
          "open": 593.82269,
          "high": 593.95166,
          "low": 583.46271,
          "close": 587.37262,
          "volume": 1689500,
          "adj_close": "587.37262"
        },
        "sort": [593.82269]
      },
      {
        "_index": "logstash-2014.07.22",
        "_type": "logs",
        "_id": "AU2qgZixPoayDyQnreXm",
        "_score": null,
        "_source": {
          "message":
["2014-07-23,593.23267,597.85266,592.50269,595.98267,1233200,595.98267"],
          "@version": "1",
          "@timestamp": "2014-07-22T23:00:00.000Z",
          "host": "packtpub",
          "path": "/opt/logstash/input/GOOG.csv",
          "date_of_record": "2014-07-23",
          "open": 593.23267,
          "high": 597.85266,
          "low": 592.50269,
          "close": 595.98267,
```

```
            "volume": 1233200,
            "adj_close": "595.98267"
          },
          "sort": [593.23267]
        }
      ]
    }
}
```

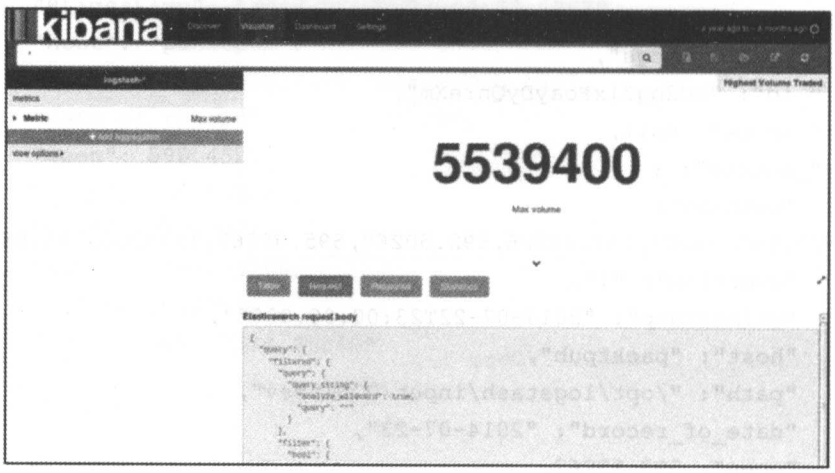
【 】



更详细的 Query DSL 相关资料请参考 Elasticsearch 的官方文档：https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html。

现在我们对于 Elasticsearch 的 Query DSL 已经有了初步了解，可以来看一下 第 2 章 构建第一条 ELK 数据管道 里的例子中 Kibana 自动创建的查询语句。

到 Kibana 可视化页面打开我们先前创建的最高成交量的度量。点击下方的箭头按钮，会调出 Request 界面，结果如下：



Kibana UI 上的 Elasticsearch 请求体

在这里可以很容易看到 Kibana 发送给 Elasticsearch 的请求体：

```
{
  "query": {
    "filtered": {
      "query": {
        "query_string": {
          "analyze_wildcard": true,
          "query": "*"
        }
      },
      "filter": {
        "bool": {
          "must": [
            {
              "range": {
                "@timestamp": {
                  "gte": 1403880285618,
                  "lte": 1419472695417
                }
              }
            }
          ],
          "must_not": []
        }
      }
    }
  },
  "size": 0,
  "aggs": {
    "1": {
      "max": {
        "field": "volume"
      }
    }
  }
}
```

以上查询语句使用查询过滤器来设置@timestamp 字段的边界区域，同时通过聚合来

获取"Volume"字段的最大值。同样的，我们也可以创建其他可视化字段。Kibana 可以创建所有类型的查询。

Elasticsearch 插件

Elasticsearch 拥有非常丰富的插件，这主要是社区在驱动。这些插件不但有助于分析集群，而且让全文结构化查询变得更简单。

让我们来看看几个插件。

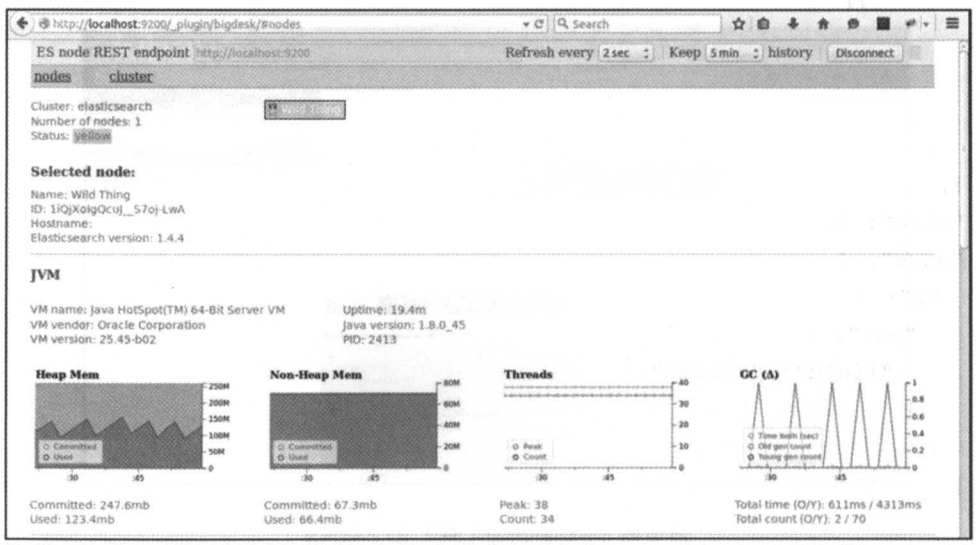
Bigdesk 插件

这个插件是 Lukas Vlcek 开发的。借助于实时图表和与 JVM、CPU、OS、主分片以及副本分片相关的各种统计信息，此插件有助于分析集群节点信息。



更信息请参考 https://github.com/lukas-vlcek/bigdesk。

下图是 Bigdesk 插件的屏幕截图：



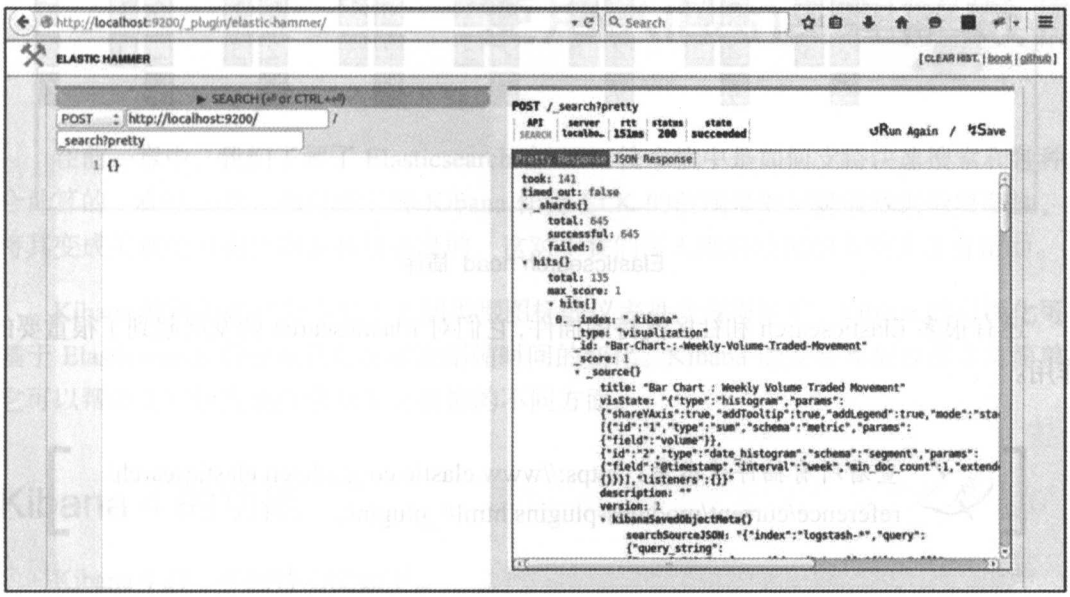
BigDesk 插件

Elastic-Hammer 插件

Elastic-Hammer 插件可以作为 Elasticsearch 的前端界面。它可以用于查询集群，并在输入查询语句时提供语法检查。



更多 Elastic-Hammer 插件信息请参考：
https://github.com/andrewvc/elastic-hammer。



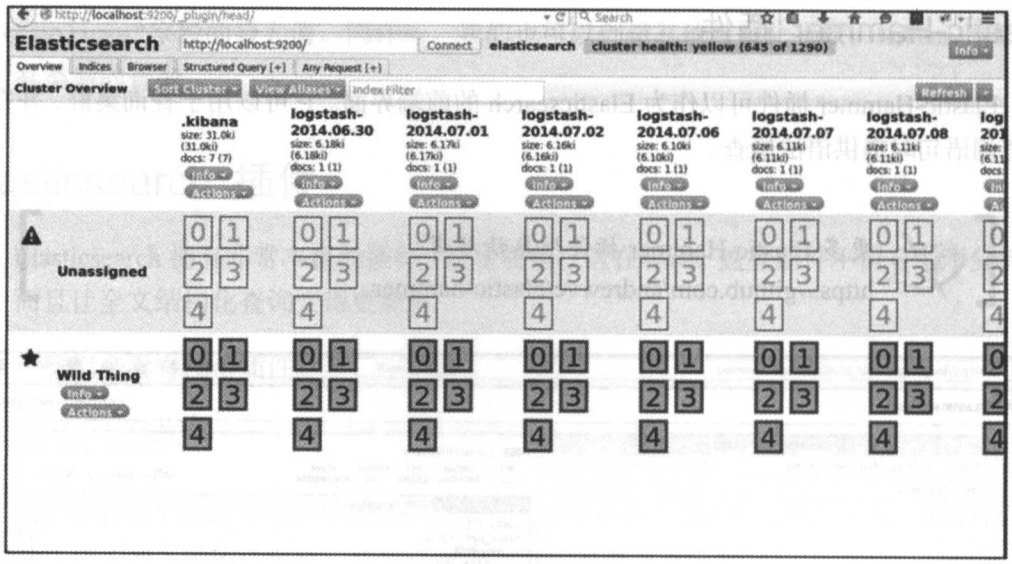
Elasticsearch Elastic-Hammer 插件

Head 插件

Head 插件能够生成集群的统计数据，并提供浏览器查询，同时还能对 Elasticsearch 索引进行结构化查询。



更多信息请参考：https://github.com/mobz/elasticsearch-head。



Elasticsearch head 插件

还有很多 Elasticsearch 和社区支持的插件，它们对 Elasticsearch 的发展起到了很重要的作用。



查看所有插件请参考：https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-plugins.html#_plugins。

总结

在这一章中，我们学习了 Elasticsearch 的基本概念，也知道了根据 Elasticsearch 索引进行查询是如何实现的，并且了解了 Kibana 如何利用 Elasticsearch 查询来有效分析索引数据，以及如何基于这些查询来实现美观的可视化图表。

在下一章，我们将更深入地学习 Kibana 各种特性的细节，以理解在 Kibana **搜索**页面是如何查询数据的。

6

使用 Kibana 理解数据

在前一章中，我们了解了 Elasticsearch 在 ELK 技术栈中是如何支持快速搜索和各种聚合计算的。在这一章，我们将了解 Kibana 作为 ELK 的前端是如何隐藏数据的复杂性，并将其变成美观的可视化图表和仪表盘的，这对于我们深入理解数据的本质非常有帮助。

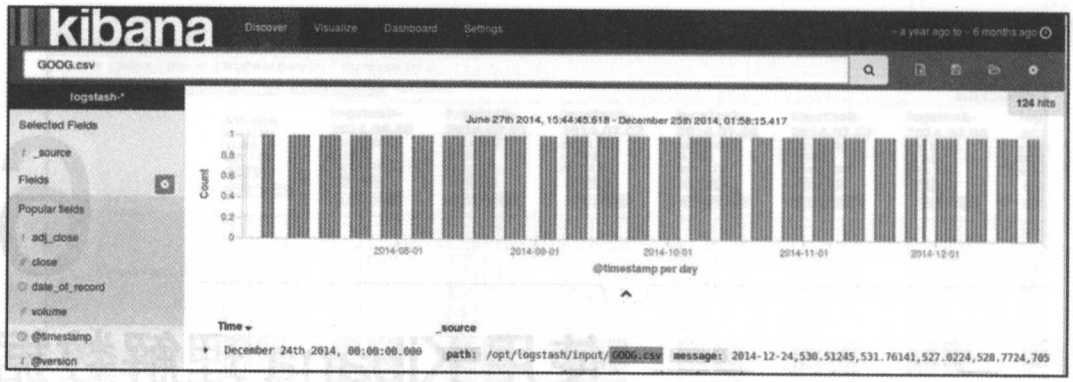
Kibana 使得创建和分享基于不同类型图标的仪表盘变得很简单。Kibana 的可视化可以基于 Elasticsearch 的查询自动显示数据随时间的变化。Kibana 的安装和配置都非常简单，它可以帮助我们快速地探索和发现数据的不同方面。

Kibana 4 的功能

Kibana 4 的一些独特功能如下。

搜索词高亮显示

在文件搜索结果列表中高亮显示搜索词：



Kibana 4 搜索词高亮显示

Elasticsearch 聚合

Kibana 4 广泛使用 Elasticsearch 的聚合和子聚合为可视化提供多种聚合功能。主要包含两种类型的聚合—分桶（Bucketing）和度量。分桶聚合会生成一系列的桶，每个桶都有一组文档，例如短语、范围、直方图等。度量聚合则计算一组文档的度量指标，例如最小值、最大值、求和，平均值等。只能在数值类型的字段上进行这样的计算。

衍生字段

衍生字段（Scripted fields）用于索引数据的动态计算。

例如，某字段需要在显示之前乘以 100，就可以将它存储为衍生字段，但衍生字段不能被搜索。

我们以下面的衍生字段作为例子：`doc['volume'].value * 100`。

这个衍生字段将始终在展示 volume 的值之前将其乘以 100。

动态仪表盘

仪表盘非常灵活，并且是动态的。可以方便地用其将各个可视化组件根据需要拖拽排列，并且数据也可以自动刷新。

Kibana 界面

Kibana 界面包含 4 个主要的标签。

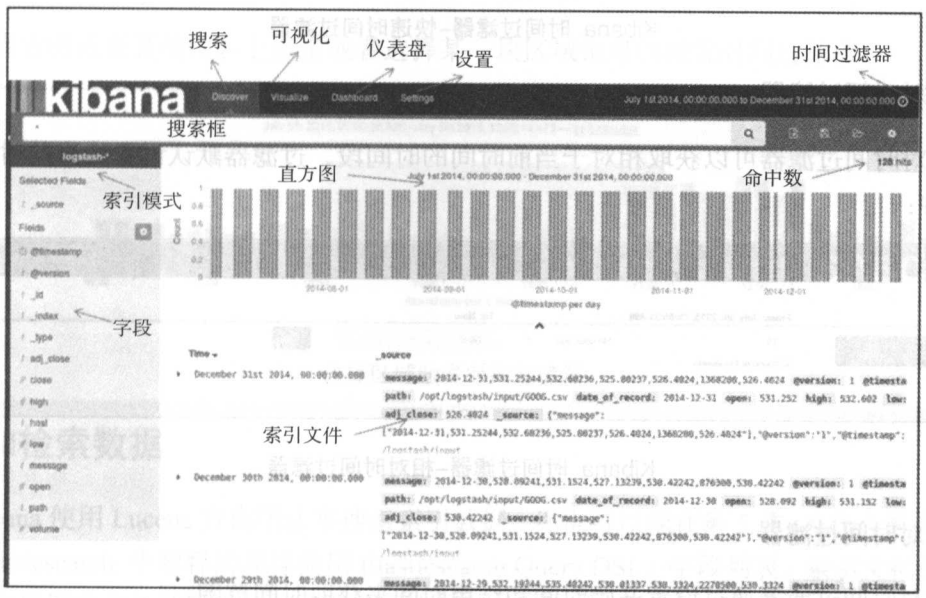
- **搜索**：在**搜索**页面可以自由搜索，或基于字段、范围等搜索。
- **可视化**：在**可视化**页面可以创建许多类型的可视化，如饼图、柱状图、折线图等，并且可以保存起来，随后在仪表盘中使用。
- **仪表盘**：**仪表盘**是多种可视化和搜索的集合，可以很简单地应用于基于点击交互的过滤器，也能基于多种数据汇总获得结论。
- **设置**：在**设置**页可以配置索引模式、衍生字段、字段的数据类型等。

接下来更详细地了解一下**搜索**页面。

搜索页面

搜索页面适用于对索引数据进行交互式搜索查询。可以做基于字段的特定搜索、过滤数据，也可以查看索引好的文档。

一个典型的 Kibana 主页，默认就是**搜索**页面，看起来如下图：



Kibana 搜索页面

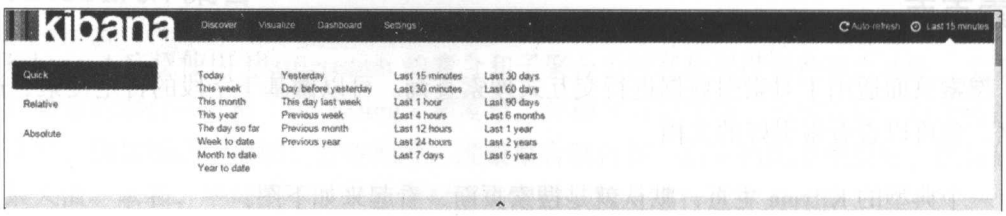
搜索页面左侧是所有的**索引模式**，顶部是**时间过滤器**和**搜索框**。同时页面上还有一个基于**@timestamp** 字段的默认**直方图**，对应搜索结果的**命中数**也显示在页面上。搜索结果默认按时间倒序显示最新的 **500** 个文档。

时间过滤器

还记得你的老板要求从数据中找出特定时间的统计数据吗？时间过滤器就适用于此类搜索。你可以过滤任何时间段的数据，这个时间段可以是**绝对时间段**，也可以是基于当前时间的**相对时间段**。对于相对时间查询，有一些快捷时间过滤器可供使用。

快捷时间过滤器

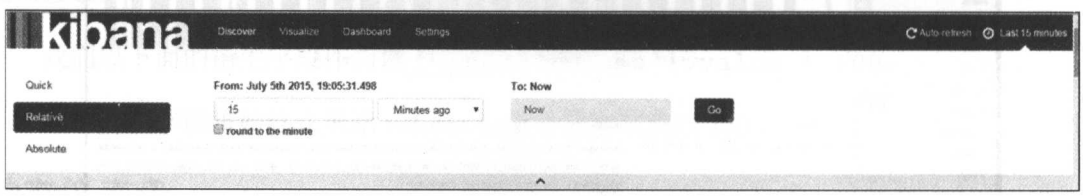
快捷时间过滤器能让我们快速获取一些时间范围：



Kibana 时间过滤器–快速时间过滤器

相对时间过滤器

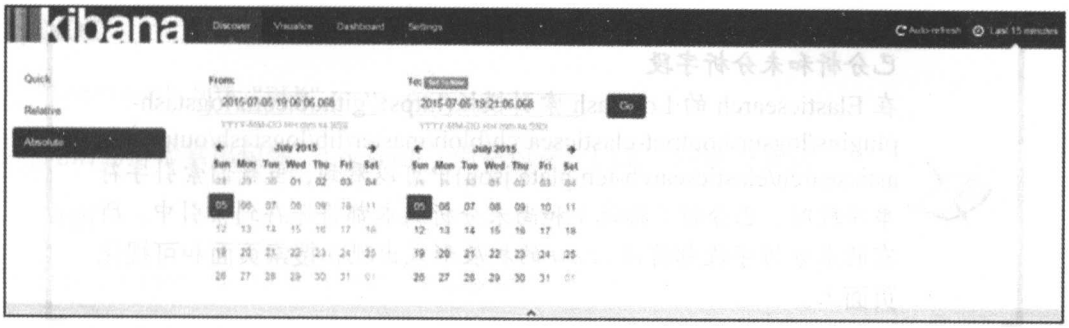
相对时间过滤器可以获取相对于当前时间的时间段，过滤器默认设置为**当前时间**的前 15 分钟：



Kibana 时间过滤器–相对时间过滤器

绝对时间过滤器

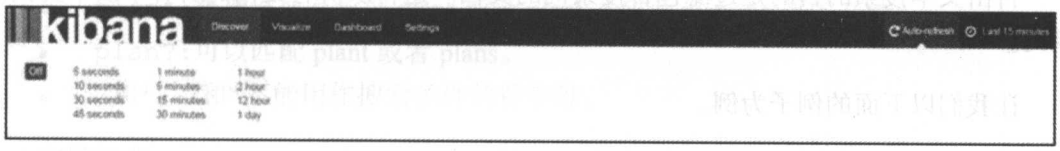
绝对时间过滤器通过设置**开始时间**和**结束时间**来获取时间范围：



Kibana 时间过滤器–绝对时间过滤器

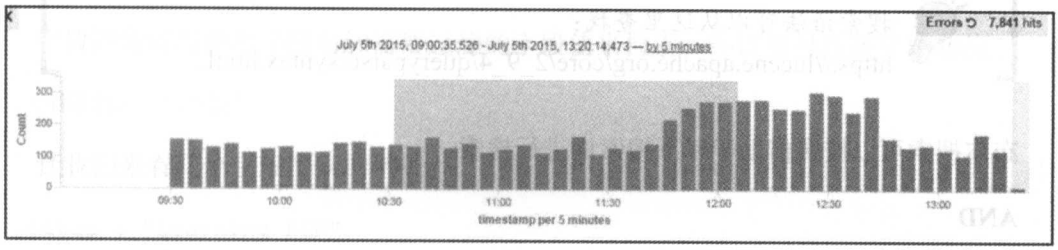
Kibana 自动刷新设置

我们可以在自动刷新设置区域设置自动刷新时间间隔：



Kibana 自动刷新设置

在直方图或者其他图表上点击或者选择某一块区域也可以触发时间过滤器：



选择区域触发时间过滤器

查询和检索数据

Kibana 使用 Lucene 查询语法来搜索索引数据。你也可以像在第 5 章 为什么需要 ELK 中的 Elasticsearch 中解释的那样使用 Elasticsearch Query DSL。字段列表、索引文档列表和直方图会根据搜索条件和时间过滤器的改变而自动刷新。



已分析和未分析字段

在 Elasticsearch 的 Logstash 索引模板(https://github.com/logstash-plugins/logstashoutput-elasticsearch/blob/master/lib/logstash/outputs/elasticsearch/elasticsearch-template.json)中可以看到，当我们索引字符串字段时，已分析（标记）和尚未分析版本都将保存到索引中。所有的未分析字段都将以.raw 的扩展形式出现在**搜索页面**和**可视化页面**上。

让我们来看一些搜索样例。

自由文本搜索

自由文本搜索的目的是过滤包含搜索词的文档。这将从所有文档的所有字段中查找搜索词。

让我们以下面的例子为例。

你想从由 packtpub 藏书集合组成的索引中检索出所有的 ELK 相关的书。那么可以在搜索框中输入“ELK”，它将会过滤出包含了“ELK”这个词的所有文件。



搜索语法可以从这里查找：
https://lucene.apache.org/core/2_9_4/queryparsersyntax.html。

布尔搜索可以按照借助于以下搜索词进行搜索：

AND

```
"Learning" AND "ELK"
```

此语句将检索出同时包含“Learning”和“ELK”的所有文档。

OR

```
"Logstash" OR "ELK"
```

此语句将检索出包含“Logstash”或者包含“ELK”的所有文档。

NOT

```
"Logstash" NOT "ELK"
```

此语句将检索出包含“Logstash”但不包含“ELK”的所有文档。

分组

```
("Logstash" OR "ELK") AND "Kibana"
```

此语句将检索出包含“Kibana”并且包含“ELK”或者“Logstash”的所有文档。

通配符搜索

你也可以使用以下字符进行通配符搜索。

- `plan*`:将搜索所有形如 plans 或者 plant 又或者 planting 等的文档。
- `plan?`:可以匹配 plant 或者 plans。
- `?`和`*`：`?`和`*`不能用作搜索条件的首字母。

字段搜索

字段搜索的目的是搜索索引文档中特定值或特定范围的字段，这些字段都显示在搜索页面的左侧。

字段搜索可以使用字段名和需要过滤的字段值来检索，并以冒号连接字段和值。

```
<字段名>: <字段值>
```

让我们来看几个字段搜索的例子：

```
title : "Learning ELK"
title : "Learning ELK" AND category : "technology"
```

范围搜索

范围搜索一般用于查询某个字段的取值范围。

比如搜索特定的日期范围：

```
date_of_record : [20140701 TO 20141231]
```

查询 volume 字段的取值在 100000 ~ 200000 之间的所有文档：

```
volume : [ 100000 TO 200000]
```

范围搜索和字段搜索可以与布尔符号组合使用，如：

```
publish_date : [20150701 TO 20151231] AND title : "Learning ELK"
```

特殊字符转义

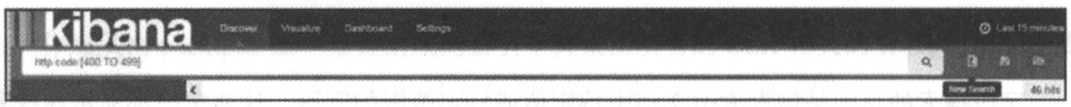
以下是特殊字符列表，如果需要在查询中使用这些特殊字符，则要使用\符号进行转义：

```
+ - && || ! ( ) { } [ ] ^ " ~ * ? : \
```

例如想要查询 `1:2`，需要转义成 `1\:2`。

新建搜索

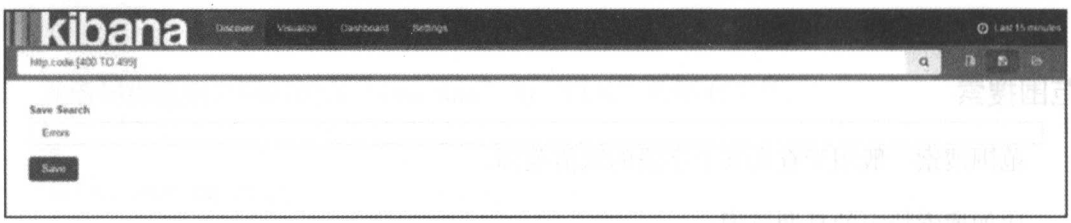
你可以点击**搜索**页面上的“New Search”按钮来打开新建搜索页面：



Kibana 新建搜索选项

保存搜索

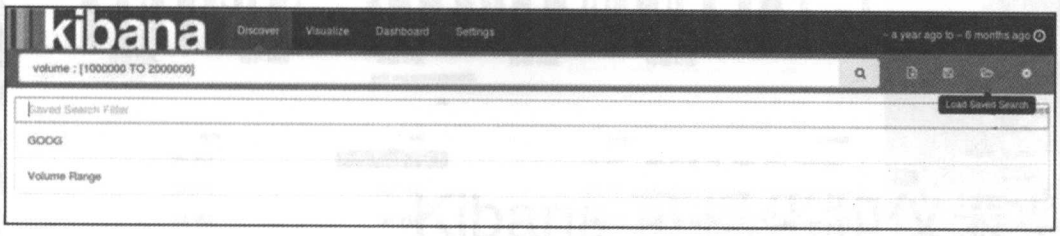
使用**搜索**页面上的“Save Search”选项可以把搜索保存起来并用于后面的可视化。已保存的搜索可以添加到仪表盘中，以便于采用传统表格方式显示信息。这对于应用程序的识别是非常重要的。



Kibana 保存搜索选项

打开已保存搜索

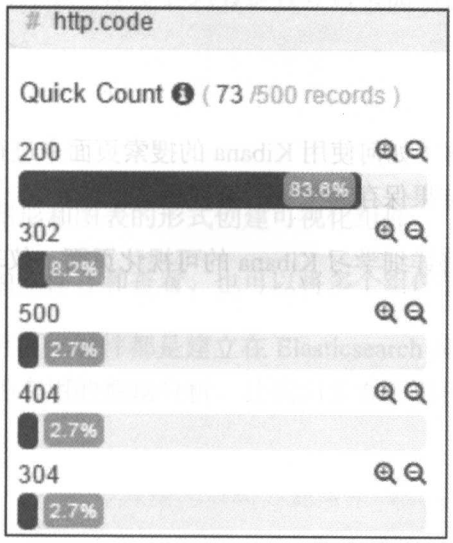
通过**搜索**页面工具栏上的“Load Saved Search”选项可以打开之前已保存的搜索：



加载已保存的搜索选项

借助字段列表来搜索字段

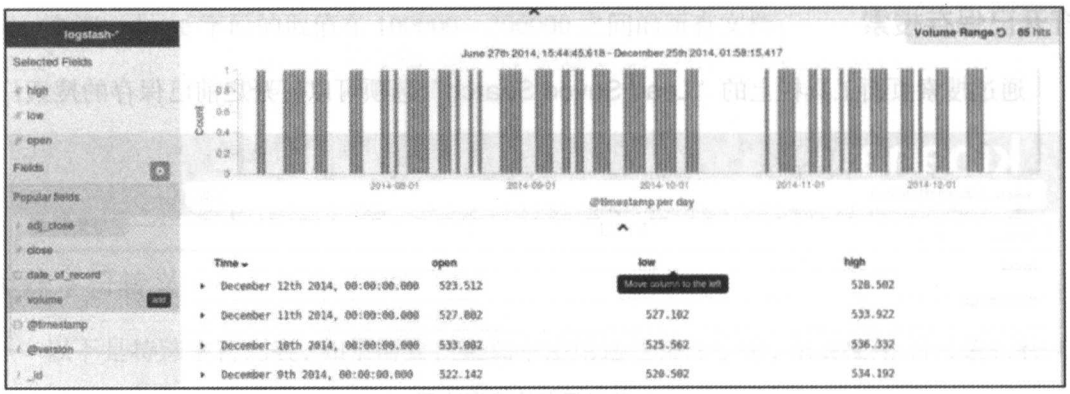
可以通过点击字段特定取值上的“正”或者“负”过滤按钮来进行字段查询。



字段列表上的放大/缩小过滤器

在上图中，如果点击“正”过滤按钮，将会过滤出 `http.code` 字段值为 200 的所有文档，如果点击“负”过滤按钮，将会过滤出 `http.code` 取值不是 200 的所有文档。

也可以通过点击左侧字段列表上字段名称旁的“add”按钮来让右侧面板显示指定的字段。这样就可以根据需要在右边的结果表中显示字段的值了。



Kibana 的字段搜索

通过这种方式，可以快速添加字段，也可以根据特定字段分类文档，还可以按照任意顺序排列字段。这对于建立快速搜索的表格非常有帮助。

总结

在这一章中，我们了解了如何使用 Kibana 的**搜索页面**来进行快速搜索，以获得对数据的理解，以及如何将搜索结果保存起来供后续使用。

下面的章节里，我们将详细学习 Kibana 的**可视化页面**、**仪表盘页面**和**设置页面**。

7

Kibana 可视化和仪表盘

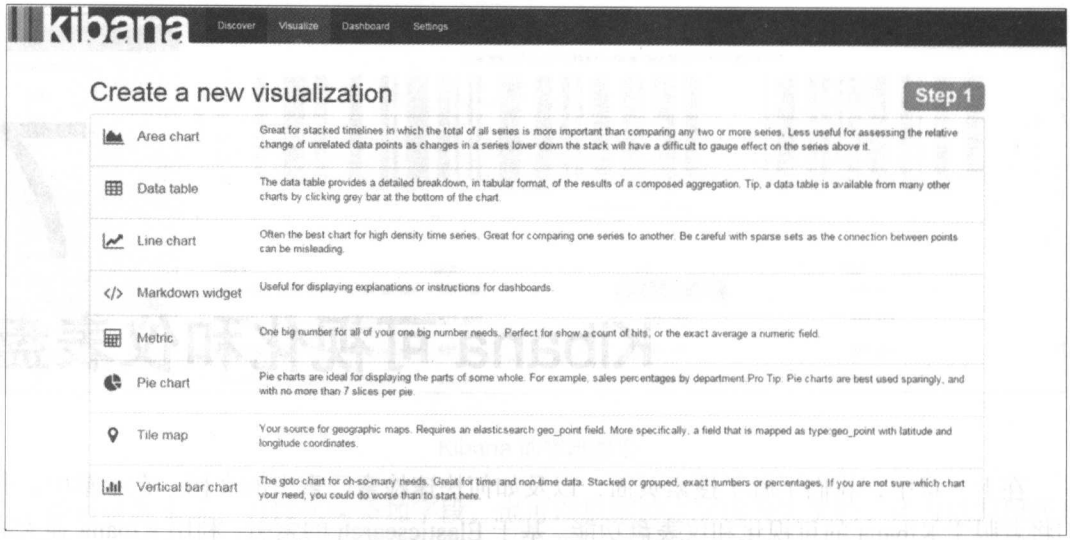
在上一章中，我们了解了**搜索**页面，以及如何快速检索已索引的文档。在本章中，我们将着眼于 Kibana 的**可视化**和**仪表盘**功能。基于 Elasticsearch 的索引，利用 Kibana 强大的能力，可以构建各种类型的图表、图形和具有多种分析功能的仪表盘。这些仪表盘可以便利地嵌入或者分享给其他人。

可视化页面

可视化页面有助于以图形和图表的形式创建可视化组件。

这些可视化组件可以独立保存和查看，也可以将多个组件放到仪表盘中使用。

在 Kibana 中，所有的可视化组件都是建立在 Elasticsearch 聚合功能的基础上的。Kibana 还支持多级聚合来进行各种有用的数据分析。让我们来看下**可视化**页面的样子：



Kibana 可视化页面

可视化页面有两个部分——创建新的可视化组件，以及从已保存列表中打开已保存的组件。

创建可视化

要创建一个新的可视化，只需点击顶部菜单栏中“Visualize”按钮，打开一个新的**可视化**页面，然后点击工具栏上的“New Visualization”按钮即可。

在**可视化**页面上创建新的可视化分三步：

1. 选择可视化类型。
2. 选择数据源（使用新建的搜索或已保存的搜索）。
3. 配置**编辑**页面上的可视化聚合属性（度量和桶）。

可视化的类型

Kibana 支持以下类型的可视化：

- 区域图

- 数据表格
- 折线图
- Markdown 小部件
- 度量
- 饼图
- 切片地图
- 垂直柱状图

在开始建立各种类型的可视化之前，先了解一下作为 Kibana 可视化基础的 Elasticsearch 聚合功能。

度量和桶聚合

度量和桶的概念来自 Elasticsearch 的聚合功能，这两个概念在 Kibana 中为数据集设计可视化的时候扮演着至关重要的角色。

桶

分桶帮助将文档分发到多个包含已索引文档子集的桶中。桶的功能与 SQL 的 GROUP BY 非常相似。分桶可以将文档根据特定的条件进行分组，然后对分组后的文档计算度量。

桶通常代表 Kibana 图表的 **X 轴**，我们也可以给桶添加子桶。

Kibana 的 **X 轴**支持如下的桶类型：

- **日期直方图**（Data Histogram）
- **直方图**（Histogram）
- **范围**（Range）
- **日期范围**（Date Range）
- **IPv4 范围**（IPV4 Range）
- **短语**（Temrs）
- **过滤器**（Filters）
- **重要短语**（Sginifiant Terms）

接下来，来了解几个重要的可视化。

日期直方图

日期直方图需要一个日期类型的字段和时间区间配置。文档将按指定的字段和时间区间分组。举个例子，如果指定**@timestamp** 字段作为**桶**，且**时间区间**为**一周**，那么文档将基于每周的数据分组，然后可以对分组后的文档计算度量，如**计数**、求**平均值**等。

直方图

直方图与**日期直方图**相似，除了要求指定的字段和区间都是数字类型的。直方图将在选定的字段上按照指定的区间对文档进行分桶。这相当于以相等区间进行范围聚合。

范围

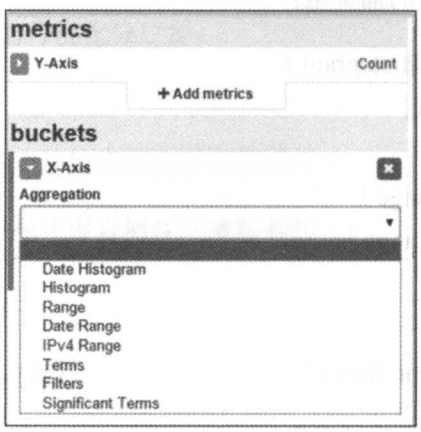
范围类似于**直方图**，但可以根据需求手动配置不同的范围。例如对于字段计数，可以选用分桶的范围为 0 ~ 1000、1000 ~ 5000 及 5000 ~ 15000 等。

日期范围

日期范围需要一个日期字段，并且为每个桶指定自定义的日期范围。

短语

短语可以用于根据任意字段的值进行文档分组，这非常类似于 SOL 中的 GROUP BY 语句。**短语**聚合还可以选择指定 Top N 或 Bottom N，或者也可以基于度量指定顺序。例如，可以根据产品类型来进行分组，并获得每个产品类型前五名。



可视化的桶

度量

度量是对每个桶中的字段的值进行计算，例如计算文档的总数、平均值、最小值或最大值。度量通常代表**区域图**、**垂直柱状图**和**折线图**的 **Y 轴**。度量的可用类型如下：

- Count（计数）
- Average（平均值）
- Sum（求和）
- Unique Count（唯一值计数）
- Min（最小值）
- Max（最大值）
- Percentile（百分比）
- Percentile Ranks（百分比等级）

让我们来了解其中几个。

Count

Count 是非常重要的度量聚合函数，它的主要目的是计算在桶聚合里每个桶中字段值的数量。

例如，要计算每一个产品类别的访问者的数量，可以指定产品类别字段为桶聚合，然后进行 count 度量聚合计算。

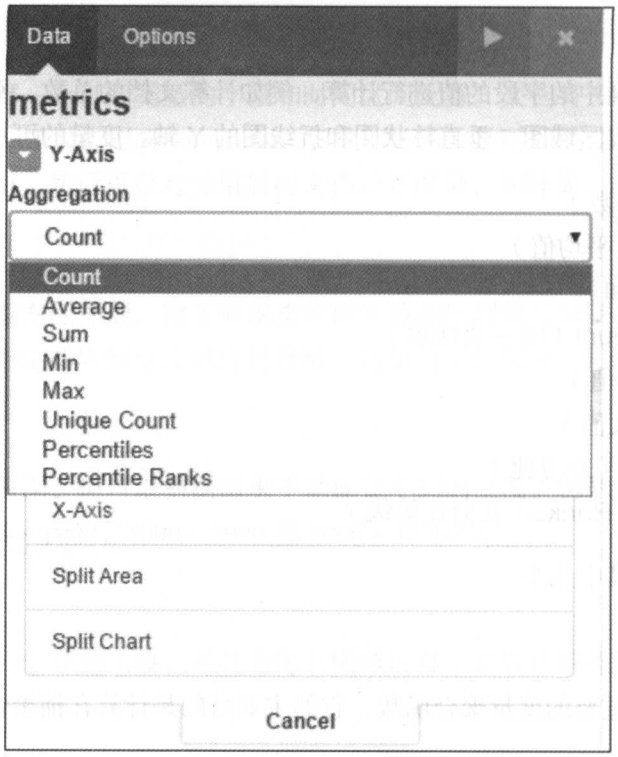
Average、Sum、Min 和 Max

类似于 Count 聚合，Average、Sum、Min 和 Max 相应地为聚合中的数字字段计算平均值、求和、最小值和最大值。

Unique Count

Unique Count 类似于 SQL 中的 `COUNT（DISTINCT fieldname）` 功能，

它能计算出字段的唯一值的数量。

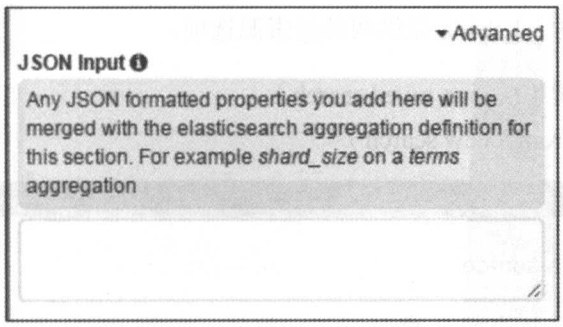


Kibana 可视化度量

高级选项

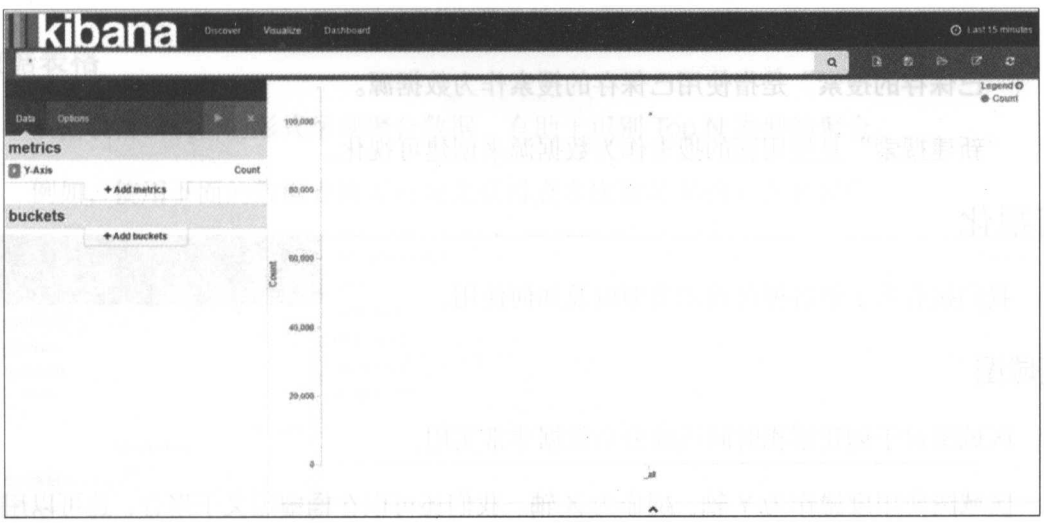
桶和度量聚合有高级选项，可以把 JSON 输入作为第 6 章 *使用 Kibana 理解数据* 中所述的衍生字段。下面的 JSON 脚本就是一个例子：

```
{ "script" : "doc['volume'].value * 100"}
```



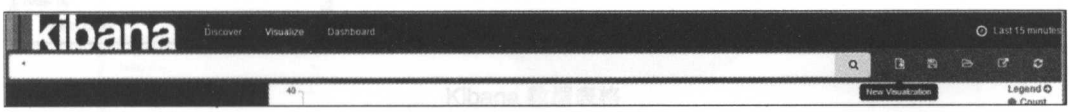
JSON 输入高级选项

总体而言，一个**新可视化**页面看起来像下图这样，工具栏在顶部，**度量**和**桶**在左侧，预览窗格在右侧：



Kibana 新建可视化页面

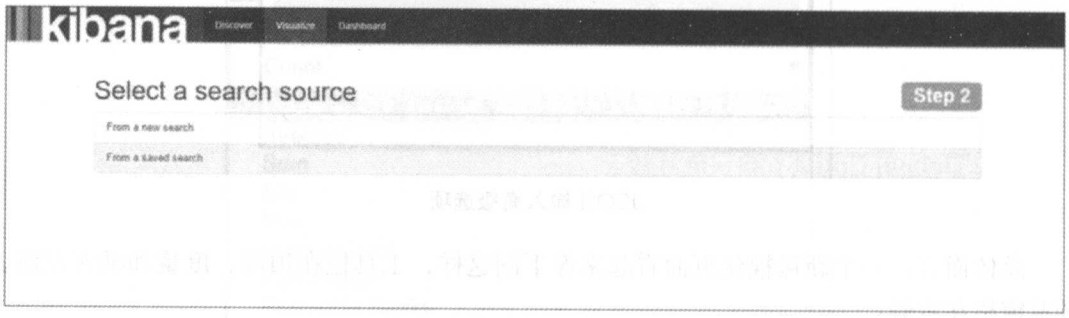
顶部的工具栏上的选项可以创建一个新的可视化、保存可视化、打开已保存的可视化、共享一个可视化和刷新可视化。



Kibana 可视化工具条

创建一个可视化时，Kibana 提供两种搜索源选项：

- **已保存的搜索**（From a saved search）
- **新建搜索**（From a new search）



Kibana 搜索源的选择

“已保存的搜索”是指使用已保存的**搜索**作为数据源。

“新建搜索”是使用新的搜索作为数据源来创建可视化。

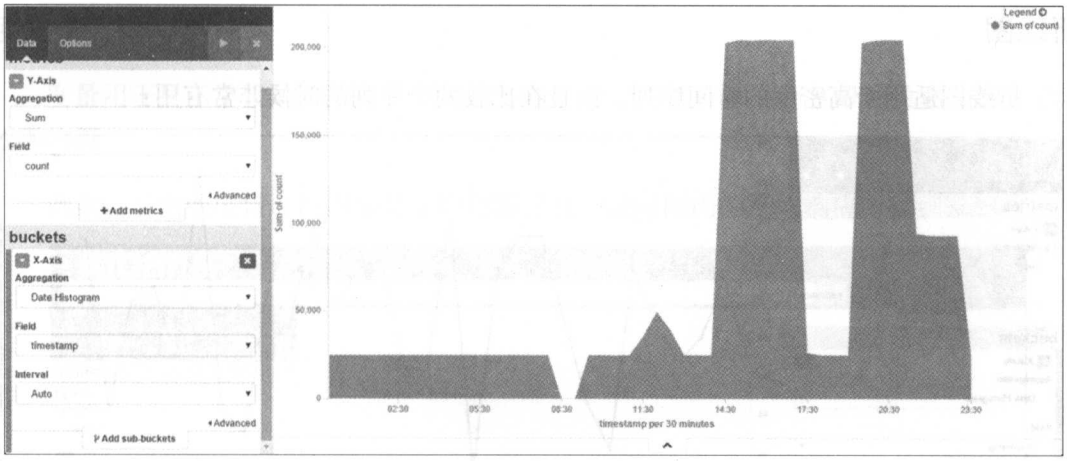
可视化

我们接着来了解各种可视化类型以及如何使用。

区域图

区域图对于创建累积时间线或分布数据非常实用。

区域图使用**度量**作为 **Y 轴**、**桶**作为 **X 轴**。我们还可以在**桶**中定义子聚合，这可以用来实现**图表分割**（Split Charts，分割成基于不同聚合的多个图表）或者**区域分割**（Split Area，分割成基于不同聚合在区域）的功能。

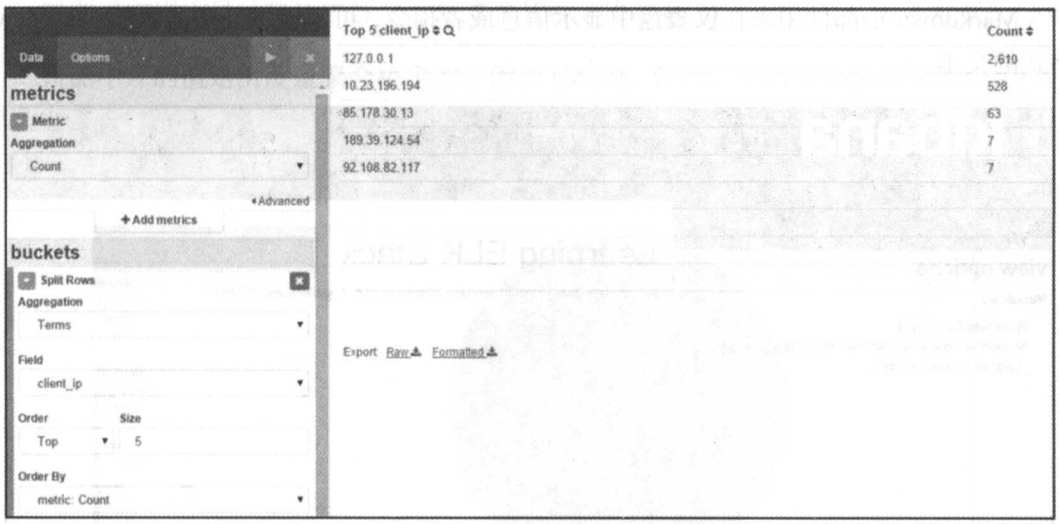


Kibana 区域图

数据表格

数据表格以表格的形式呈现聚合数据，有助于识别 Top N 类型的聚合。

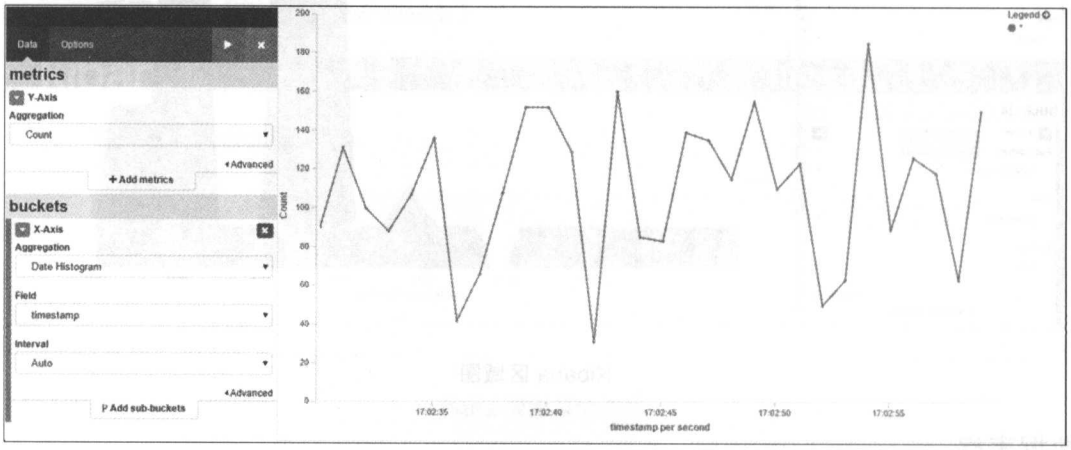
例如，使用下面的数据表格可视化来获得点击次数最多的前五名客户：



Kibana 数据表格

折线图

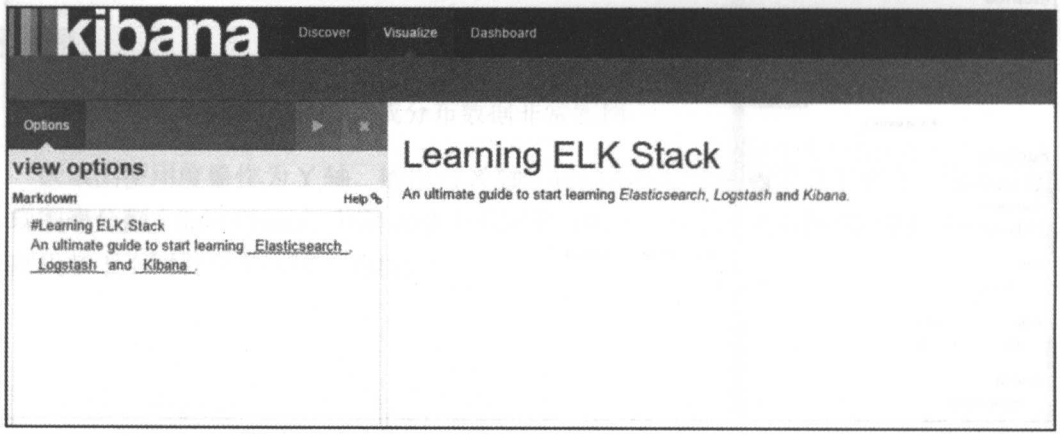
折线图适用于高密度的时间序列，而且在比较两个序列的时候非常有用：



Kibana 折线图

Markdown 小部件

Markdown 小部件用来在**仪表盘**中显示信息或者指令，可以显示任意需求的 Markdown 格式的文本。

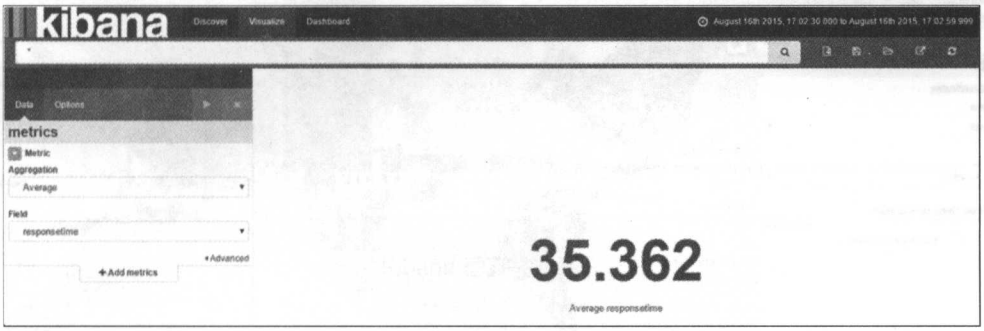


Kibana 的 Markdown 小部件

度量

度量用于显示字段的单个数字类型的分析。可以用来计算一个字段的总命中数、总和或平均值。

例如，下面的度量可以用来显示应用程序在一段时间内的平均响应时间：



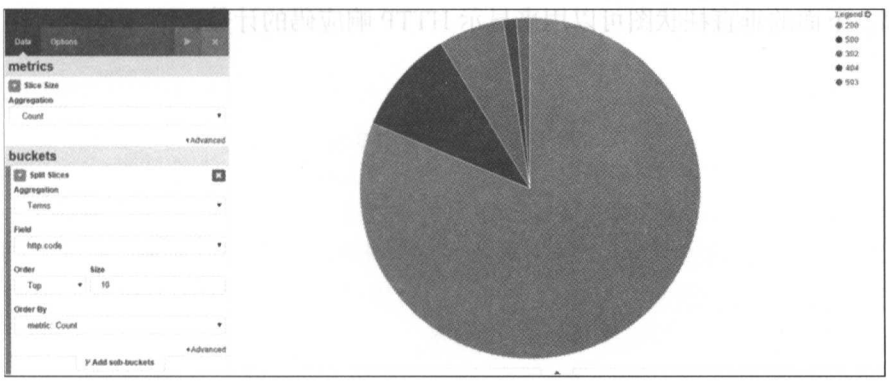
Kibana 度量

饼图

饼图通常用于显示整体中各个部分或者其百分比关系。饼图中的片代表了数据的分布。

饼图中片的值是由度量聚合决定的，例如 Count、Sum，或者 Unique Count。桶聚合则定义了图表中的数据类型。

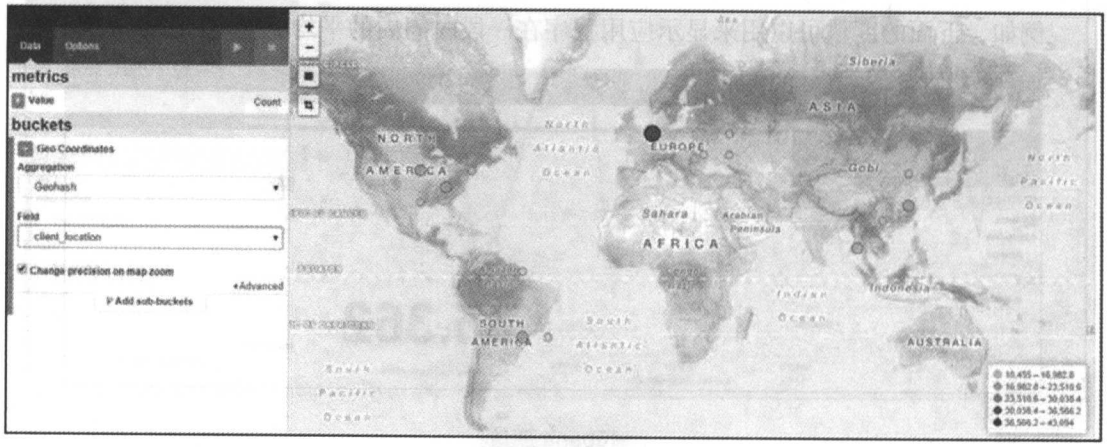
例如，下面的饼图可以用来显示应用程序的不同响应码的分布：



Kibana 饼图

切片地图

*切片地图用来根据geo坐标定位地理位置。*这是基于Geohash桶聚合实现的，Geohash聚合会将多组坐标分组到一个桶中。



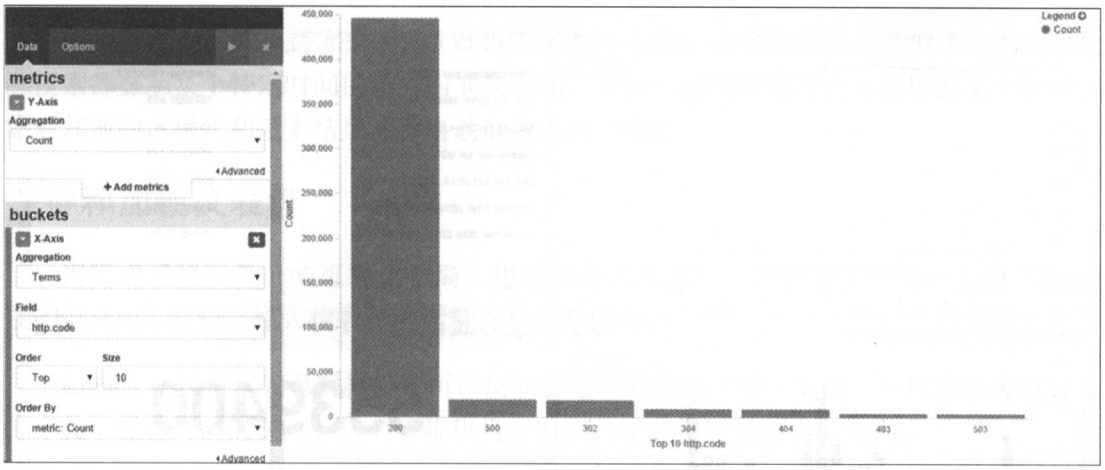
Kibana 切片地图

垂直柱状图

垂直柱状图可被用于多种场景。垂直柱状图对基于时间和非时间的字段都表现得很好。垂直柱状图可以是单独的柱状图，也可以是累积柱状图。

Y轴是度量，X轴是桶聚合。

例如，下面的垂直柱状图可以用来显示HTTP响应码的计数：



Kibana 垂直柱状图

仪表盘页面

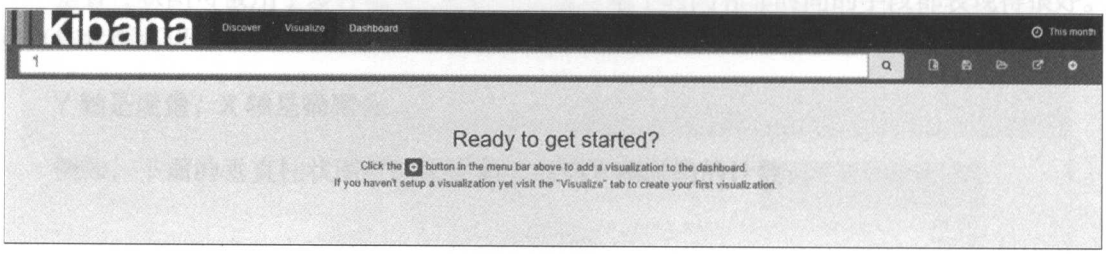
Kibana **仪表盘**就是一组可以按任意顺序排列的已保存可视化或者已保存搜索的集合。可视化可以在多个仪表盘中使用，如果可视化有更新，会自动反映到所有用到的仪表盘中。仪表盘可以被很简单地保存和共享。

让我们来看下**仪表盘**页面的样子：



创建新的仪表盘

第一次点击顶部 Kibana **仪表盘**页面的链接的时候，会显示一个空 Kibana 仪表盘，等待添加可视化：



Kibana 新仪表盘界面

点击工具栏最右边带圆圈的“+”按钮，可以添加任何已保存可视化或者在**搜索**页面以表格形式保存的搜索。添加完可视化之后，还可以根据自己喜好对其移动、编辑或删除。也可以调整其大小或者根据需要进行拖放操作。

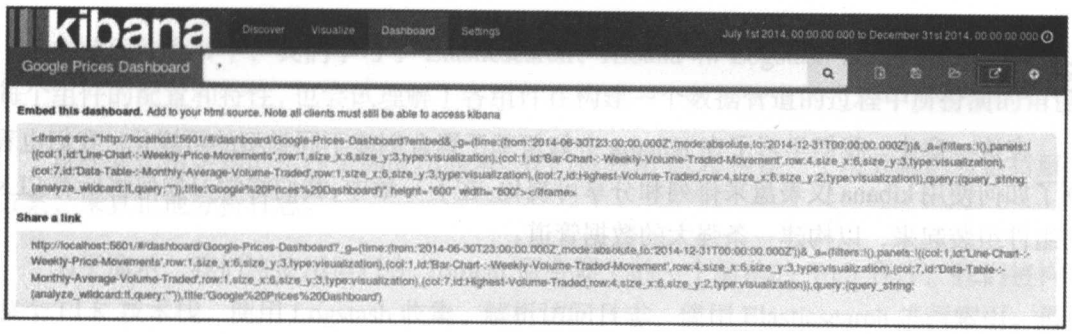
在顶部的时间过滤器中设置刷新间隔后，仪表盘会自动刷新以显示最新值。

Kibana 仪表盘还支持点击过滤（click to filter）功能，这在需要仪表盘中所有的可视化和搜索都显示某个特定时间的事件时非常有用。例如，你可以点击直方图里的某个柱体，然后其他的可视化和搜索结果都将自动地做出相应变化。

保存和加载仪表盘

当完成了仪表盘上的可视化布局，想要保存仪表盘时，只需单击工具栏上的“Save Dashboard”按钮，输入仪表盘的名称然后保存即可。

Kibana 在保存仪表盘时提供了可以反映特定时间值的工具。为此，在保存仪表盘时有一个关于保存时间的选项。这对于提供特定时间的系统快照是非常有用的。

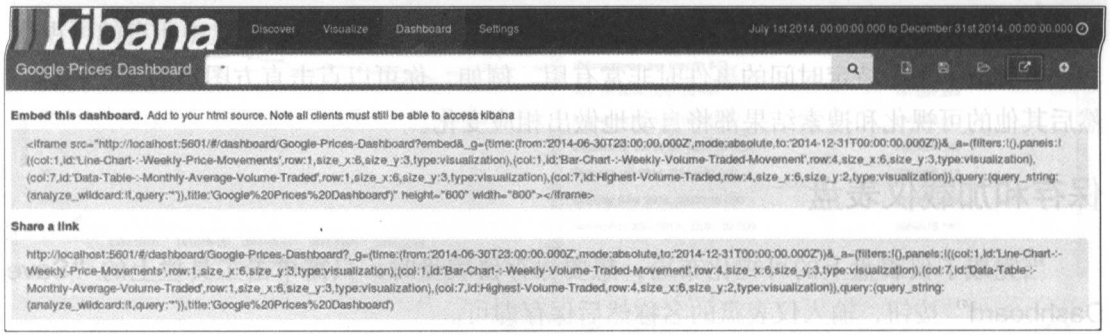


Kibana 保存仪表盘

点击工具栏上的“Load Dashboard”按钮，然后在已保存的仪表盘中进行选择，就能加载已保存的仪表盘。

分享仪表盘

设计完仪表盘并保存后，可以通过链接来分享，也可以通过 Iframe 标签嵌入到其他应用程序中。只需要点击工具栏上的“Share”按钮，就会显示一段可以嵌入到其他应用程序的代码，以及直达该仪表盘的链接。代码和链接都可以复制和共享。



Kibana 分享界面

总结

在这一章中，我们学习了在 Kibana 中如何基于聚合创建不同类型的可视化。我们也看到了如何使用 kibana **仪表盘**来排列和分享可视化。在下一章中，我们将学习如何将所有 ELK 的组件组装起来，以构建一条强大的数据管道。

8

构建完整的 ELK 技术栈

在前面的章节中，我们学习了 Elasticsearch、Kibana 和 Logstash 的基础知识。熟悉了每个组件的配置和特性，也尝试理解了各组件在构建一个数据管道的过程中所扮演的角色。

现在要把所学到的东西全部应用起来。我们将利用 ELK 构建一个端到端的可运行解决方案，来真正地分析日志。

为了演示，我们将使用运行在 Tomcat 服务器上的一个简单的 Web 程序。我们将构建整个 ELK 技术栈，使用 Logstash 收集、解析访问日志，使用 Elasticsearch 进行索引。最后在 Kibana 界面中看到各种搜索和展示效果。

输入数据集

输入数据集是下面这种格式的持续的 Tomcat 访问日志流：

```
10.0.0.2 - - [08/Sep/2015:17:39:46 +0100] "GET /elk/demo/10 HTTP/1.1" 200 40
10.0.0.2 - - [08/Sep/2015:17:39:47 +0100] "GET /elk/demo/11 HTTP/1.1" 200 39
10.0.0.3 - - [08/Sep/2015:17:39:48 +0100] "GET /elk/demo/12 HTTP/1.1" 200 39
10.0.0.2 - - [08/Sep/2015:17:39:49 +0100] "GET /elk/demo/13 HTTP/1.1" 200 39
10.0.0.2 - - [08/Sep/2015:17:39:50 +0100] "GET /elk/demo/14 HTTP/1.1" 200 39
10.0.0.4 - - [08/Sep/2015:17:39:51 +0100] "GET /elk/demo/15 HTTP/1.1" 200 40
10.0.0.2 - - [08/Sep/2015:17:39:52 +0100] "GET /elk/demo/16 HTTP/1.1" 200 39
10.0.0.2 - - [08/Sep/2015:17:39:53 +0100] "GET /elk/demo/17 HTTP/1.1" 200 39
10.0.0.5 - - [08/Sep/2015:17:39:54 +0100] "GET /elk/demo/18 HTTP/1.1" 200 39
10.0.0.2 - - [08/Sep/2015:17:39:55 +0100] "GET /elk/demo/19 HTTP/1.1" 200 39
```

```
10.0.0.2 - - [08/Sep/2015:17:39:56 +0100] "GET /elk/demo/20 HTTP/1.1" 200 40
10.0.0.6 - - [08/Sep/2015:17:39:57 +0100] "GET /elk/demo/21 HTTP/1.1" 200 38
10.0.0.2 - - [08/Sep/2015:17:39:58 +0100] "GET /elk/demo/22 HTTP/1.1" 200 40
10.0.0.2 - - [08/Sep/2015:17:39:59 +0100] "GET /elk/demo/23 HTTP/1.1" 200 39
```

上面这种日志格式是一种常见的 Apache 日志格式，是在 Tomcat 的 `conf` 目录下的 `server.xml` 文件中定义的，如下：

```
    <Valve className="org.apache.catalina.valves.AccessLogValve"
directory="logs"
               prefix="localhost_access_log." suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
```

日志模式的格式如下：

```
  %h  %l  %u  %t  "%r"  %s  %b
```

- `%h`：表示远程主机名（或 IP 地址）。
- `%l`：表示远程逻辑用户名。
- `%u`：表示已认证的远程用户。
- `%t`：指定日期和时间。
- `%r`：表示请求。
- `%s`：表示返回 HTTP 码。
- `%b`：表示响应发送的字节数，不包括 HTTP 头。

配置 Logstash 输入

在本节中，我们将配置 Logstash 从 Tomcat 的访问日志中读取数据，并在 Elasticsearch 中为日志创建索引，过程中还会根据 grok 模式对日志进行过滤和字段提取。

访问日志的 Grok 表达式

正如我们已知的，Logstash 安装包中已经包含了一些常用 grok 表达式。

Logstash 的常用 grok 表达式列表可以在 GitHub 上查看：

https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns。

Logstash 自带的 Apache 通用日志格式的 grok 模式如下：

```
COMMONAPACHELOG %{IPORHOST:clientip} %{USER:ident} %{USER:auth}
\[%{HTTPDATE:timestamp}\] "(?:%{WORD:verb} %{NOTSPACE:request}(?:
HTTP/%{NUMBER:httpversion})?|%{DATA:rawrequest})"
%{NUMBER:response} (?:%{NUMBER:bytes}|-)
```

所以，可以在 Logstash 中直接使用 COMMONAPACHELOG 作为匹配模式来处理如下的输入消息：

```
input{
file{
path =>"/var/lib/tomcat7/logs/localhost_access_logs.txt"
start_position =>"beginning"
}
}
```

接下来需要为输入消息指定 grok 模式来进行匹配，为消息分配时间戳字段，并根据需要转换某些字段的数据类型：

```
filter{
 grok {
      match => { "message" => "%{COMMONAPACHELOG}" }
    }
date{
    match => ["timestamp","dd/MMM/yyyy:HH:mm:ss Z"]
}
mutate{
convert => ["response","integer"]
convert => ["bytes","integer"]
}
}
```

最后，配置输出插件把过滤后的消息发送到 Elasticsearch，这里我们使用默认的 9200 端口：

```
output{
elasticsearch {
host => "localhost"
```

```
}
}
```

现在每部分的配置都清楚了，让我们看一下针对 Tomcat 日志文件的完整配置：

```
input{
file{
path =>"/var/lib/tomcat7/logs/localhost_access_log.txt"
start_position =>"beginning"
}
}

filter{
 grok {
      match => { "message" => "%{COMMONAPACHELOG}" }
   }

date{
    match => ["timestamp","dd/MMM/yyyy:HH:mm:ss Z"]
}
mutate{
convert => ["response","integer"]
convert => ["bytes","integer"]
}
}

output{
elasticsearch {
host => "localhost"
}
}
```

现在，使用这个配置来启动 Logstash：

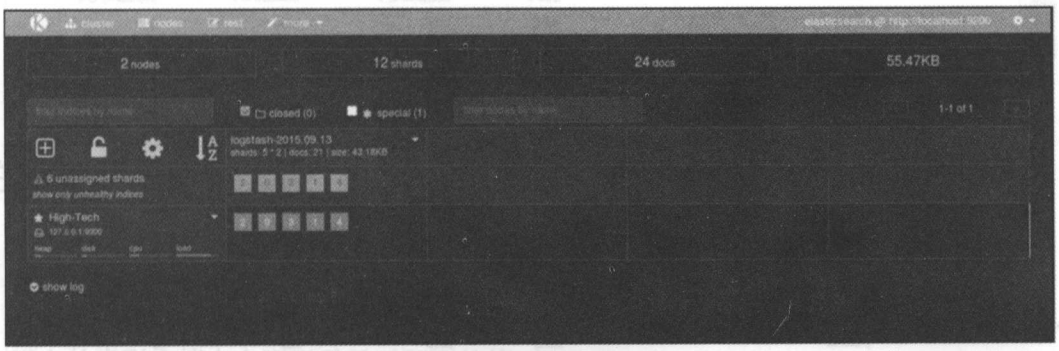
```
$ bin/logstash -f logstash.conf
```

Logstash 会按照定义好的配置运行，并持续将输入事件处理并保存到 Elasticsearch 的索引中。可以在控制台看到类似下面的输出：

```
May 31, 2015 4:04:54 PM org.elasticsearch.node.internal.InternalNode
start
INFO: [logstash-4004-9716] started
Logstash startup completed
```

现在可以在 Elasticsearch 中看到 Apache 访问日志。Logstash 能够解析输入的每一条 Apache 访问日志数据，并根据 grok 模式将一条数据拆分成不同的架构化字段。然后就可以很容易地针对 HTTP 响应码、请求方法和不同的 URL 做分析。

到了这一步，就可以打开之前在第 1 章 ELK 技术栈介绍中安装好的 Elasticsearch Kopf 插件控制台，来验证一下文档是否已经被索引好，也可以查询下这些文档。



Elasticsearch Kopf 界面

如果能在 Elasticsearch 中看到一些 Logstash 的索引，就证明 Logstash 配置运行正常。

Kibana 可视化

现在已经验证了数据在 Elasticsearch 中索引成功，我们可以去看看 Kibana 的展示界面，从数据中得到一些有用的分析。

运行 Kibana

正如第 1 章 ELK 技术栈介绍所述，可以从 Kibana 安装目录下启动服务：

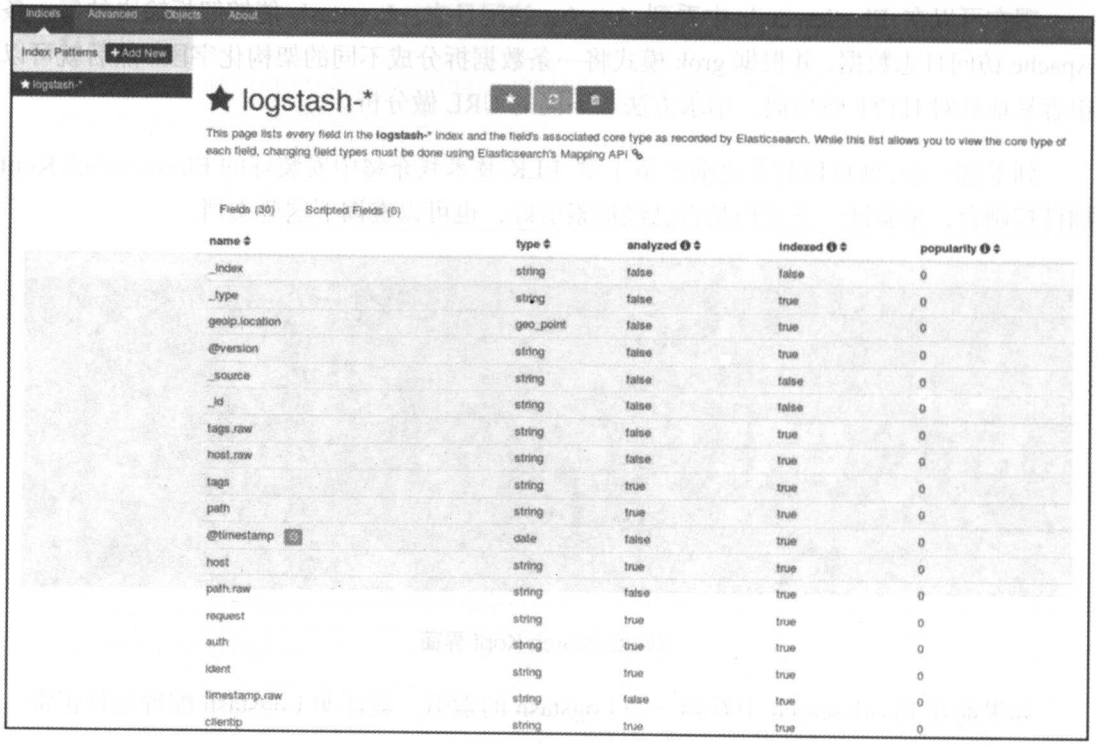
```
$ bin/kibana
```

现在，用浏览器打开下面的 URL，如果 Kibana 正常启动运行，会有一个类似下面的屏

幕截图：

```
http://localhost:5601
```

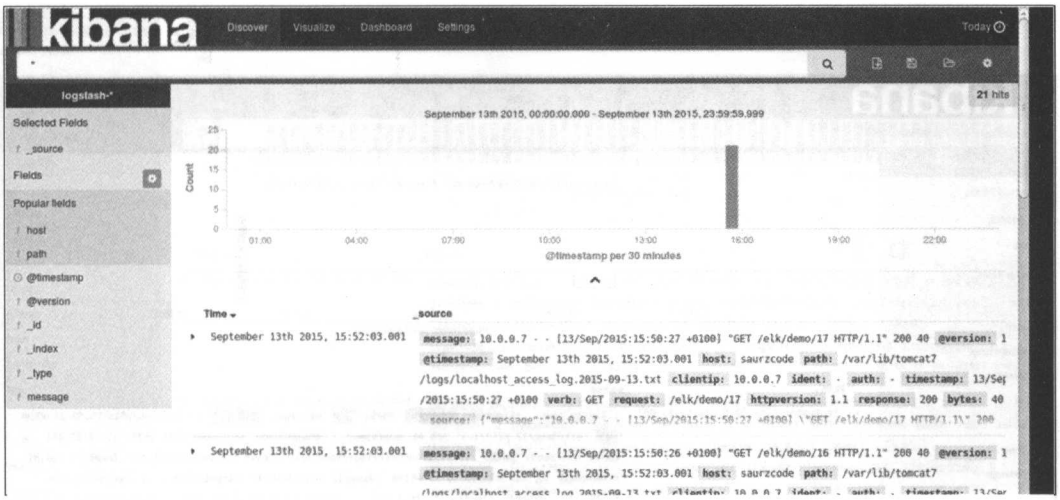
可以在 Kibana 的**设置**页面的 indice 标签页来验证我们的索引和字段：



Kibana 设置页

这个页面显示了所有被索引的字段及其数据类型、索引状态和人气值。

因为我们已经设置好 Kibana 默认采用 `logstash-*`索引，它开始显示索引数据的直方图统计，以及 JSON 格式的字段相关联数据，如下所示：

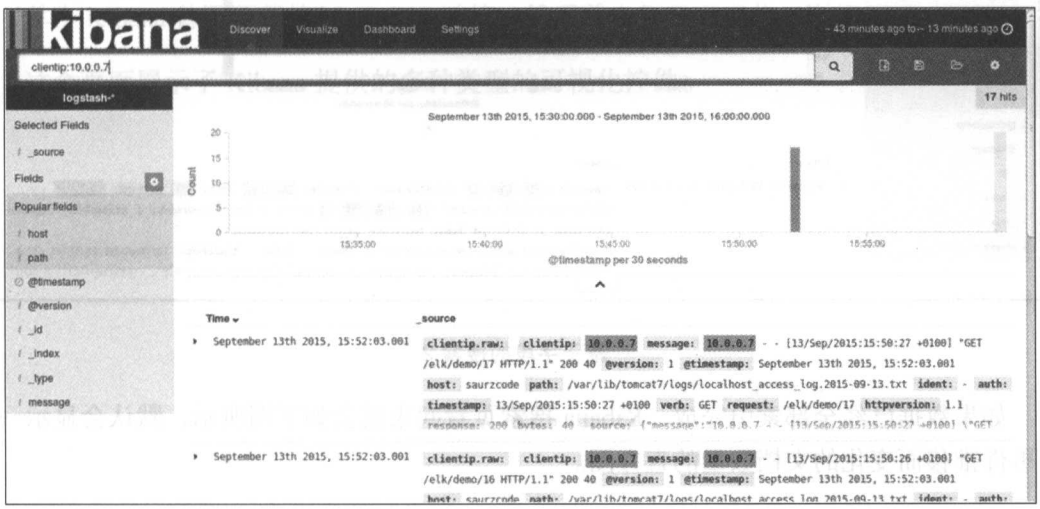


Kibana 搜索页显示的索引结果

在搜索页进行搜索

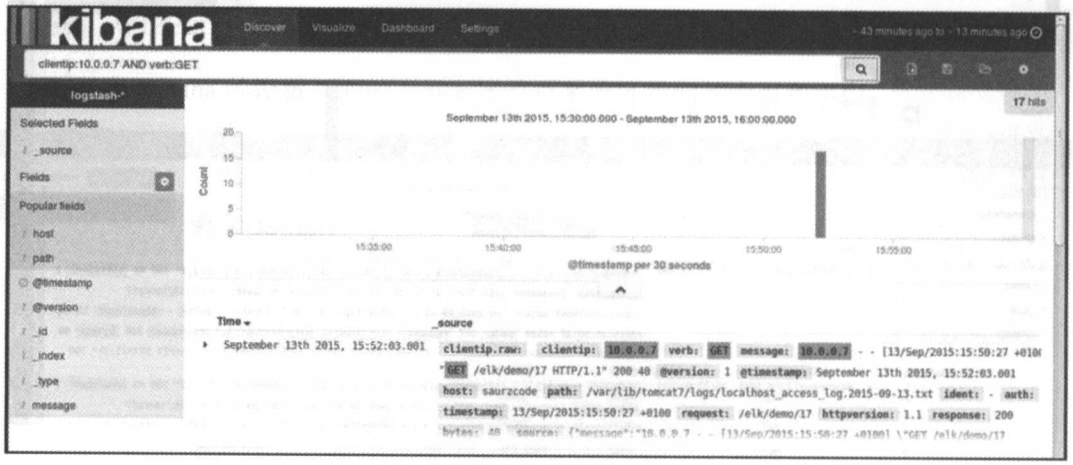
数据一旦被索引，我们就可以对字段进行快速搜索，以分析一些数据。

要查找一个指定的客户端 IP，我们可输入 `clientip:10.0.0.7` 这样的搜索命令，页面上的索引文档中会高亮显示匹配到的值：



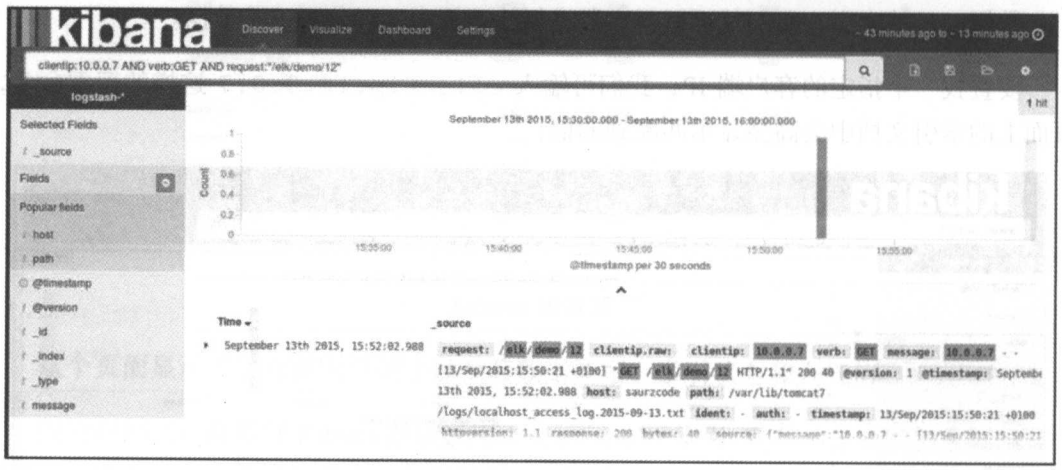
在搜索页中执行基于字段的搜索

要查找指定的客户端 IP 发出的所有 GET 请求，可以执行如下的搜索：



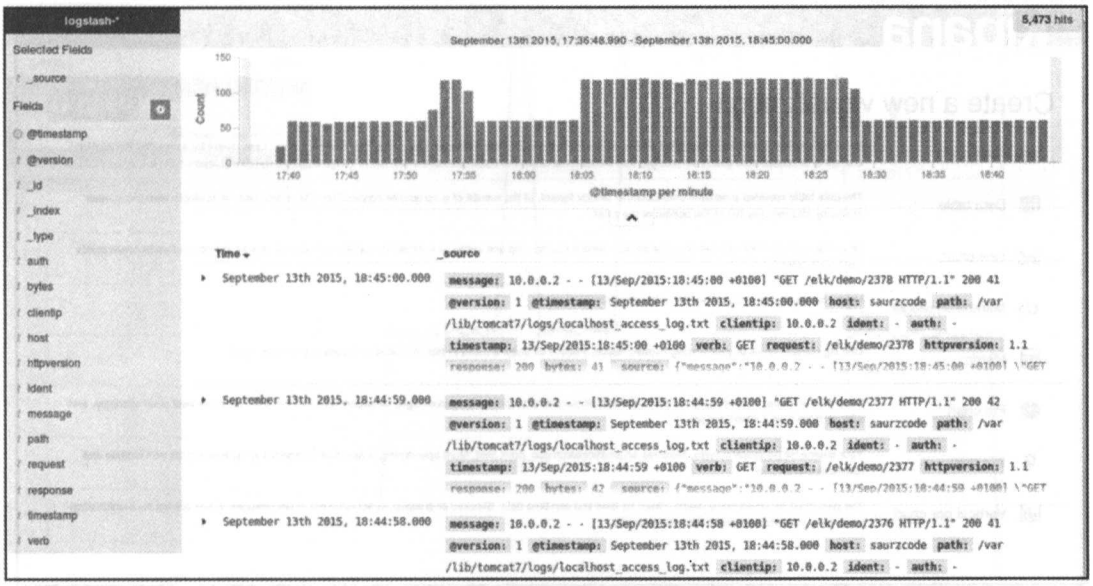
基于字段的搜索 1

要查找指定客户端 IP 的某个特定 GET 请求，可以执行如下图所示的搜索：



基于字段的搜索 2

如果数据已经全部索引完成，Kibana **搜索**页看起来就会如下图所示，默认会显示一个随事件推移而变化的文档数量的直方图。



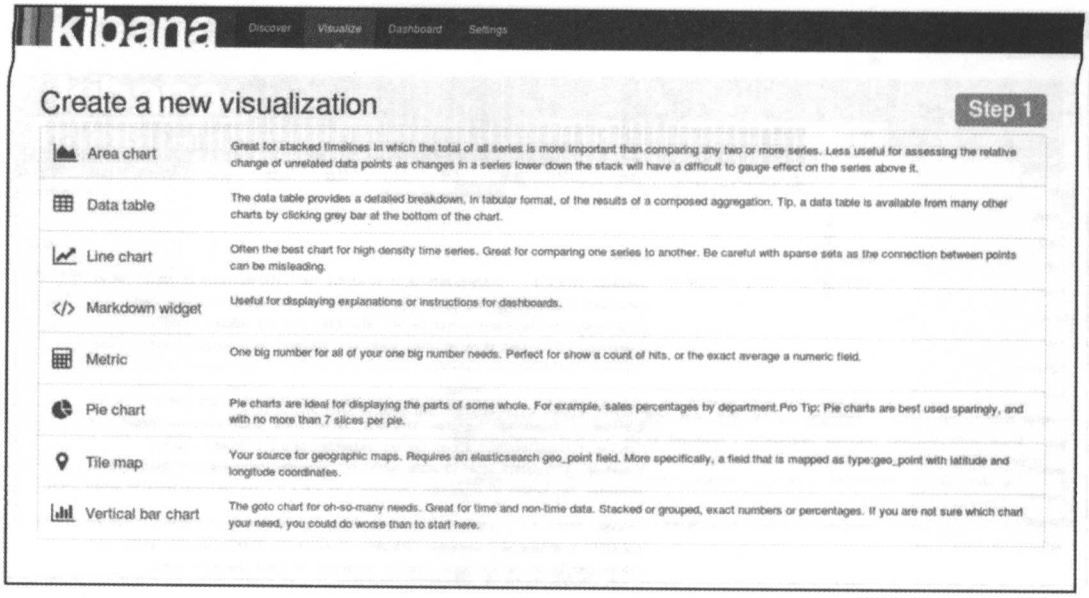
全部索引完成后的 Kibana 搜索页

可视化—图表

让我们在 Kibana 可视化页创建一些基本的可视化组件，以便稍后在仪表盘中使用。

单击 Kibana 首页顶部的 Visualize 链接，然后单击新建可视化的图标。

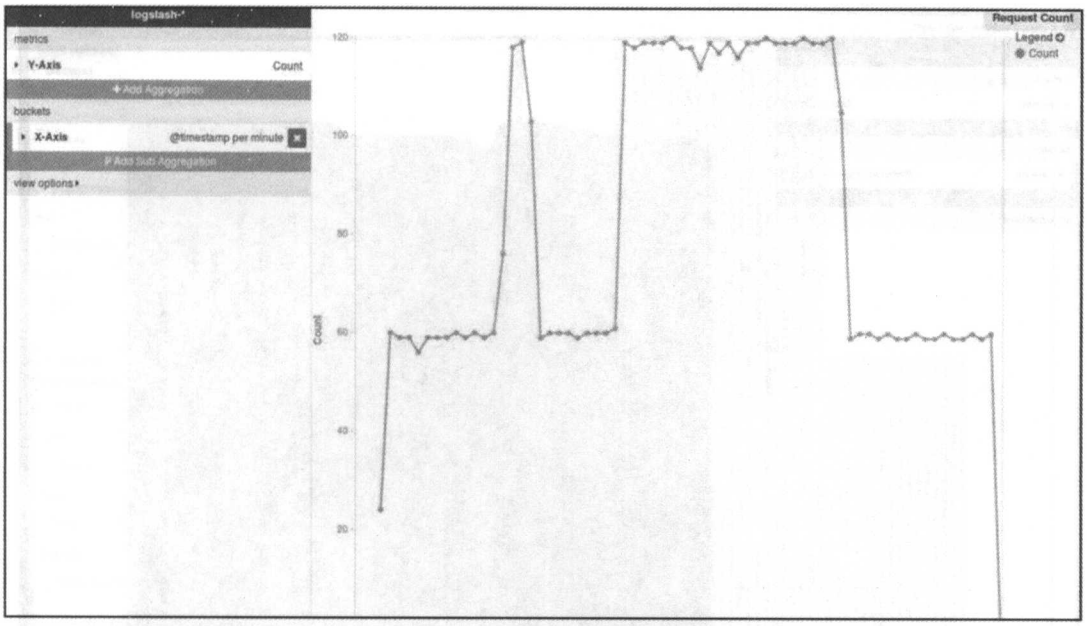
这个页面展示了 Kibana 提供的多种类型的可视化控件：



Kibana 可视化菜单

创建折线图

我们将创建的第一个可视化是**折线图**，显示应用程序随着时间的推移的点击数。为此，我们设置 **Y 轴的度量**为“Count”，**X 轴的桶**为“Date Histogram”，然后单击“Apply”。生成的**折线图**如下：

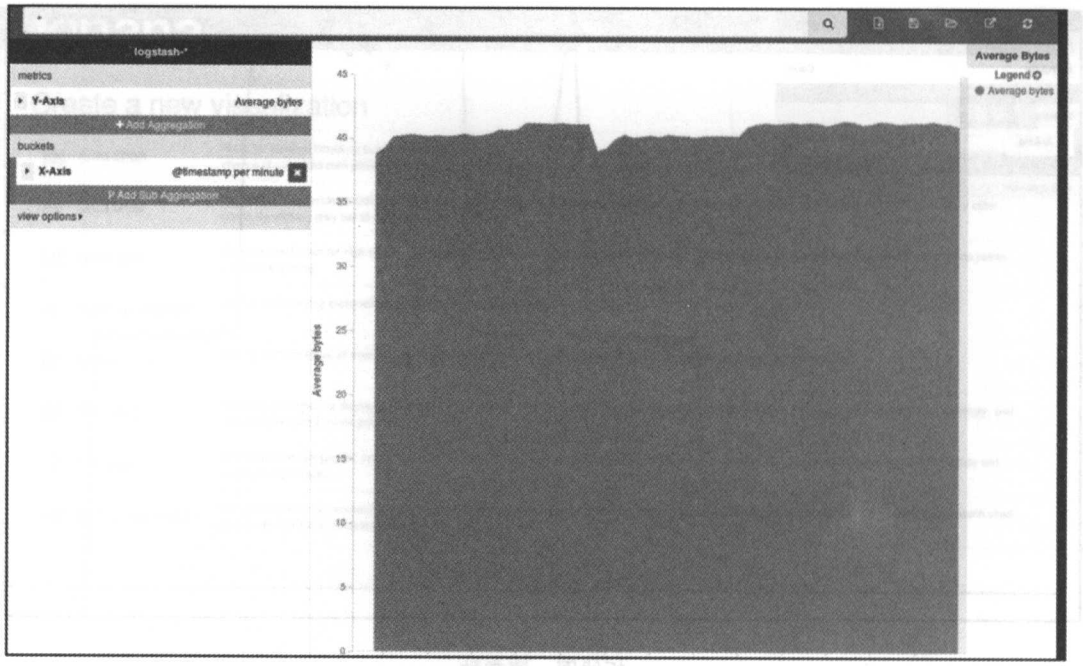


折线图—请求数

现在，可以将**折线图**以任意名字保存，稍后将在**仪表盘**拉取使用。

创建区域图

我们可以创建一个随时间推移而变化的已传输字节数的**区域图**，如下所示。为此，我们设置 Y 轴的**度量**为“Average”，设置 Field 为“bytes”。生成的**区域图**如下：

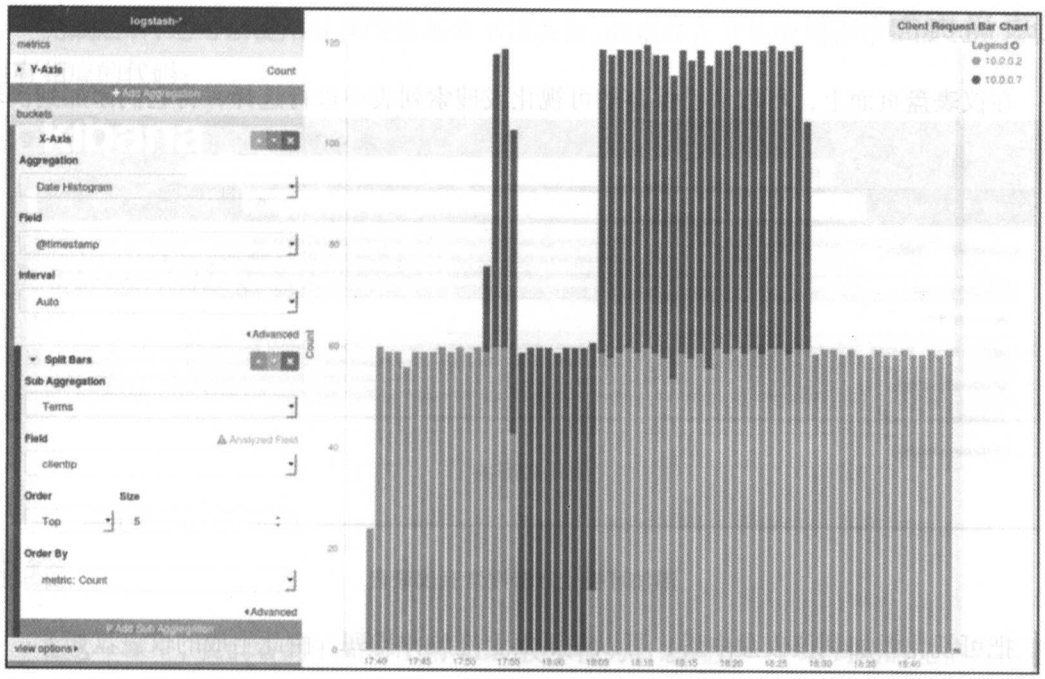


区域图—平均字节数

现在，可以将**区域图**以任意名字保存，稍后将在**仪表盘**拉取使用。

创建柱状图

我们将创建一个拆分的垂直柱状图，用来显示多个客户端上的请求数。设置 Y 轴的度量为“Count”，设置 X 轴的聚合为“Date Histogram”。通过 Split Bars 特性来使用子聚合，可以使用 clientip 来进行拆分：

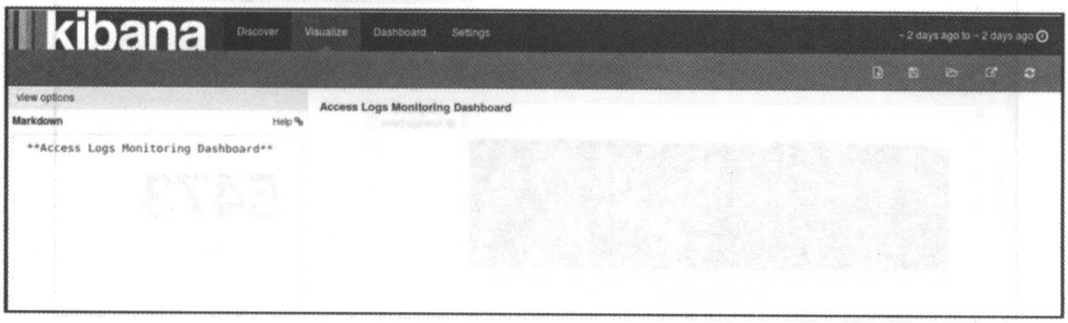


柱状图—每客户的请求数

现在，可以将**柱状图**以任意名字保存，稍后将在**仪表盘**拉取使用。

创建 Markdown

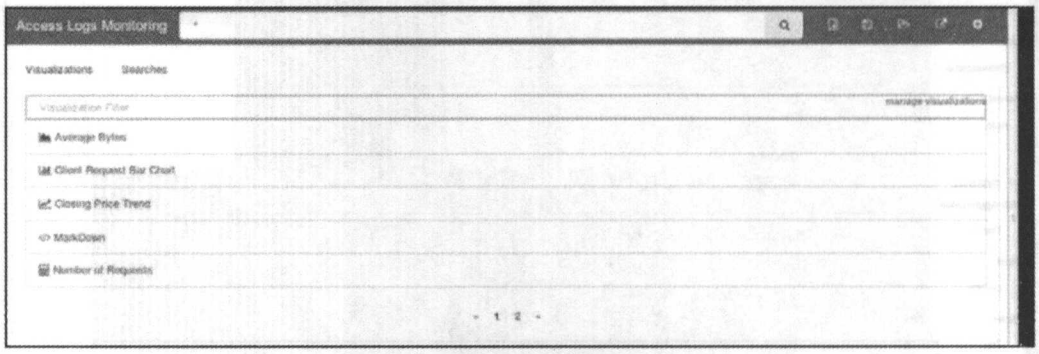
Markdown 是轻量级的标记语言，语法简单，常用于各种文档。我们将创建一个 Markdown 来描述**仪表盘**。



Markdown

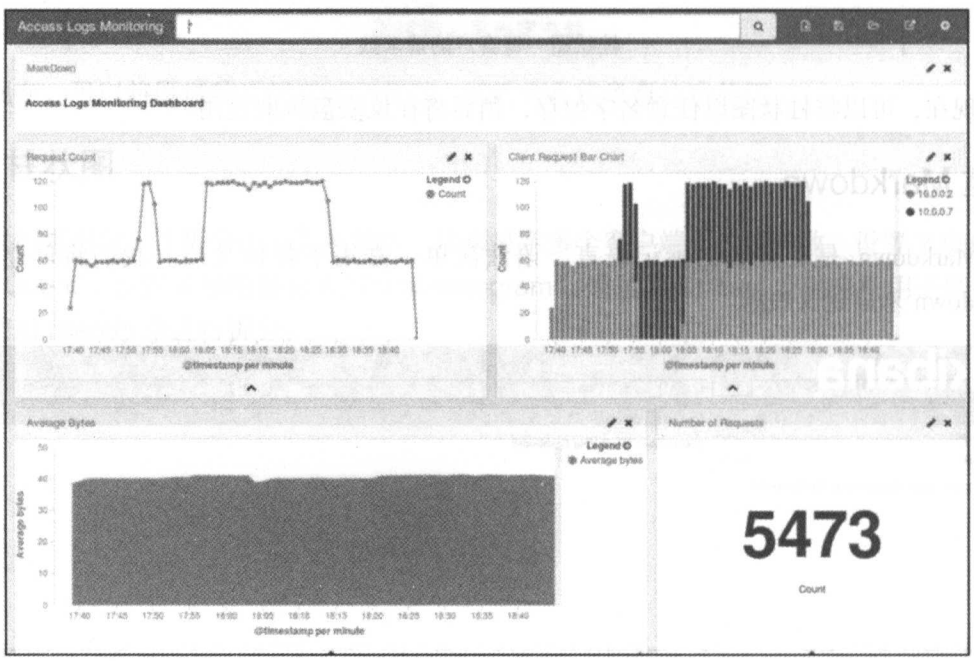
仪表盘页面

在仪表盘页面上，可以从已保存的可视化或搜索列表中进行选择，将它们添加到仪表盘中：



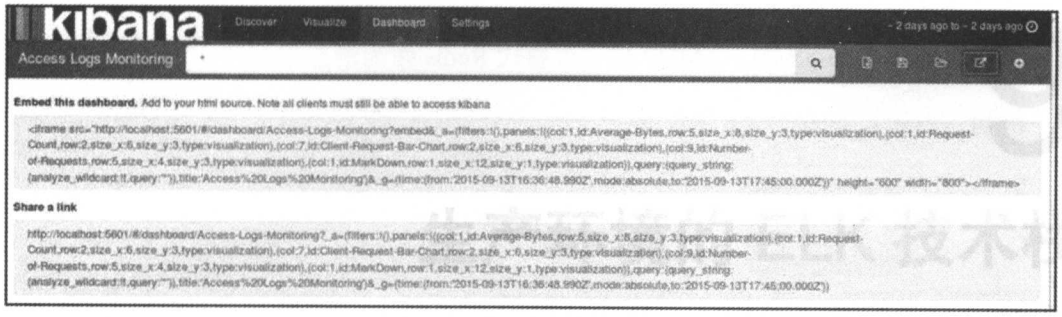
添加可视化或者搜索到仪表盘

把可视化添加到仪表盘中后，可以在页面上拖放并排列。由此生成的仪表盘如下：



仪表盘—访问日志监控

完成之后，还可以使用分享按钮来分享仪表盘，如果要在其他应用程序中嵌入仪表盘，也有相应的代码：



共享仪表盘

总结

在这一章中，我们看到了如何使用 ELK 技术栈为日志构建一个端到端的完整数据管道，这有助于我们从数据中得到有用的分析。本章帮助我们了解将 Elasticsearch、Logstash 和 Kibana 整合到一起的相关特性，以帮助构建我们自己的分析管道。

在下一章中，我们将看到 ELK 技术栈在生产环境中的实际案例，以及它带来的帮助。

9

生产环境的 ELK 技术栈

到目前为止，我们看到了如何使用 ELK 技术栈来找出日志中有价值的信息，并为应用程序的多个数据源建立一个集中的日志解决方案。

在端到端的日志管道中，我们都是在本地机器上配置 ELK 的，使用的是 Elasticsearch、Logstash 和 Kibana 的本地实例。

在本章中，我们将看看如何在有着大量数据和各种数据源的生产环境中使用 ELK 技术栈。一些大型公司，如 Bloomberg、LinkedIn、Netflix 等，已经成功地在生产环境使用 ELK 技术栈，使得 ELK 技术栈得以普及。

当我们说到生产级别实施 ELK 技术栈时，有一些隐含的前提条件：

- 防止数据丢失。
- 数据保护。
- 可扩展性的解决方案。
- 数据保留。

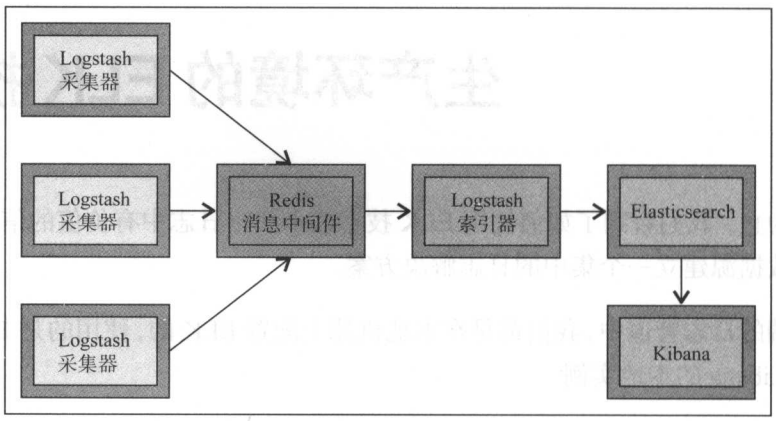
防止数据丢失

对于生产系统来说，防止数据丢失是非常关键的，因为监控和调试在很大程度上取决于系统中每一条日志事件都能保存下来。否则，整个分析或者调试系统都可能会失效，然后止步于系统中失去的一些重要事件。

在 Logstash 的索引器之前引入一层消息中间件可以防止数据丢失。消息中间件（如

Redis）在处理大型数据流时被证明是非常有效的，因为 Logstash 将数据索引到 Elasticsearch 时可能会变得很慢。在 Logstash 忙于将数据索引到 Elasticsearch 的情况下，Redis 可以很好地帮助缓冲数据。如果索引失败，它还提供了一层保护，事件会保存在消息队列中而不会丢失。ZeroMQ、RabbitMQ、AMQP 也可以替代 Redis 作为消息中间件。

例如，下面的架构会很有用：



使用了消息代理的 ELK 架构中使用消息中间件

数据保护

由于数据价值巨大，而且携带大量的机密信息，因此在 ELK 技术栈的各个环节，数据保护都是非常重要的。Elasticsearch 的索引必须防止未经授权的访问，Kibana 仪表盘也应该保护起来。我们也可以设置一个 nginx 的反向代理来访问 Kibana 实例，这样就可以通过用户名和密码来访问 Kibana 控制台。

Kibana 支持对请求做 SSL 加密，包括客户端到 Kibana 服务端的请求，以及 Kibana 服务端到 Elasticsearch 的请求。

为了对浏览器和 Kibana 服务之间的通信进行加密，可以在 `kibana.yml` 中配置 `ssl_key_file` 和 `ssl_cert_file` 两个参数。

以下是对从 Kibana 服务器发出的请求进行加密的 SSL（PEM 格式）：

- `ssl_key_file: /path/to/your/server.key`

- `ssl_cert_file: /path/to/your/server.crt`

Elasticsearch shield 可以用来为 Elasticsearch 中的数据提供索引级别的访问控制。我们可以在 shield 中为 Kibana 创建一个角色，并确定赋予 Kibana 用户某些访问权限，配置如下。

```
kibana4:
  cluster:
      - cluster:monitor/nodes/info
      - cluster:monitor/health
  indices:
    '*':
      - indices:admin/mappings/fields/get
      - indices:admin/validate/query
      - indices:data/read/search
      - indices:data/read/msearch
      - indices:admin/get
     '.kibana':
      - indices:admin/exists
      - indices:admin/mapping/put
      - indices:admin/mappings/fields/get
      - indices:admin/refresh
      - indices:admin/validate/query
      - indices:data/read/get
      - indices:data/read/mget
      - indices:data/read/search
      - indices:data/write/delete
      - indices:data/write/index
      - indices:data/write/update
      - indices:admin/create
```

我们也可以给予 Kibana 服务器级别的角色，允许其访问 `.kibana` 索引，配置如下：

```
kibana4_server:
  cluster:
      - cluster:monitor/nodes/info
      - cluster:monitor/health
  indices:
```

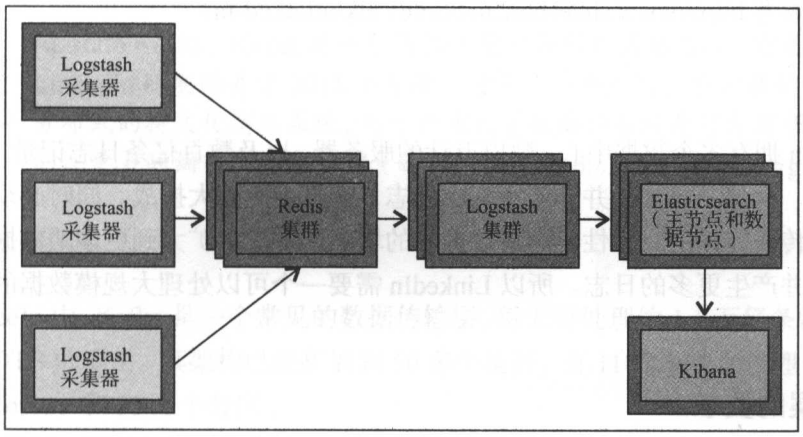
```
'.kibana':
  - indices:admin/create
  - indices:admin/exists
  - indices:admin/mapping/put
  - indices:admin/mappings/fields/get
  - indices:admin/refresh
  - indices:admin/validate/query
  - indices:data/read/get
  - indices:data/read/mget
  - indices:data/read/search
  - indices:data/write/delete
  - indices:data/write/index
  - indices:data/write/update
```

注意：shield 不是免费的，而是 Elastic 提供的收费服务的一部分。Search Guard 是另外一个免费工具，也能很好地保护 Elasticsearch 的安全。更多的细节在 http://floragunn.com/searchguard 中可以找到。

系统可扩展性

随着应用程序数据量的增长，日志分析系统要有良好的可扩展性。此外，有时系统的负载会突然升高，更需要日志分析系统来分析应用的具体情况。ELK 技术栈提供了按照需要简单扩展每个组件的能力。

可以随时在集群增加更多的 Elasticsearch 节点（主节点和数据节点）。对于大集群，建议有三个主节点（一主两备）。此外，对于大数据量的搜索和索引需求，可以增加负载均衡或路由节点。此外，可以增加更多的 Logstash 和 Redis 实例，也可添加多个 Kibana 实例。一个典型可扩展的架构如下：



数据保留

由于 Elasticsearch 不可能保存所有的数据，所以在构建日志分析系统时，制定数据保留策略是非常重要的，否则可能导致数据丢失。因此需要有一个能自动删除超过一定时间的旧索引的程序。

Elasticsearch Curator(https://github.com/elasticsearch/curator)可用来帮助管理索引数据。可以使用 Curator 定时删除不需要的旧索引。例如，下面的命令可以在指定的时间删除超过 10 天以上的索引文件，可以用 crontab 设置每天运行：

```
curator --host 10.0.0.7 delete indices --older-than 10 --time-unit days \
--timestring '%Y.%m.%d'
```

ELK 技术栈实施案例

ELK 社区已经相当庞大，并且还在迅速成长，所以得到了越来越多的关注。让我们来看一些成功的 ELK 技术栈实施案例。

LinkedIn 的 ELK 技术栈

LinkedIn 是一家面向企业的社交网站，它主要用于职业网络。LinkedIn 在 2003 年 5 月 5 日推出。截至 2015 年 3 月，LinkedIn 在 200 多个国家和地区有超过 3 亿 6400 万个用户。

具体可参考 http://www.slideshare.net/tinle1/elk-atlinked-in。

问题描述

LinkedIn 拥有多个数据中心、数以万计的服务器，以及数百亿条日志记录。每天记录、索引、搜索、存储、可视化并分析所有的日志，这是一个巨大挑战。同时必须维护访问控制、存储和传输方面的安全性。随着数据量的增长，系统将扩大到更多的数据中心、更多的服务器，并产生更多的日志。所以 LinkedIn 需要一个可以处理大规模数据的高效的日志分析管道。

对解决方案的要求

LinkedIn 需要的日志分析解决方案必须满足以下要求：

- 要能水平扩展，这样可以在需要时添加更多的节点。
- 处理速度要快，尽可能接近实时。
- 成本要低。
- 具备灵活性。
- 有一个庞大的社区支持。
- 开源。

解决方案

ELK 技术栈符合以上所有的标准。目前 LinkedIn 的许多团队都在使用 ELK。当前 LinkedIn 内部 ELK 技术栈的使用状况如下。

- ELK 集群数量超过 100 个，分布在 6 个数据中心，有 20 多个团队在使用。
- 一些大的集群：
 - 超过 320 亿个文档（超过 30TB）。
 - 平均每天索引 30 亿个文档（约 3 TB）。

目前 LinkedIn 在 ELK 技术栈的架构中使用了 Elasticsearch、Logstash、Kibana 和 Kafka。



Apache Kafka：Kafka 是一个高吞吐量的分布式消息系统，它是由 LinkedIn 研发的并于 2011 年开源。它是一个快速的、可扩展的、分布式的持久化消息系统，对于产生大量数据的系统是非常有帮助的。更多的细节可以在 Kafka 的网站找到 http://kafka.apache.org。

LinkedIn 的 Kafka

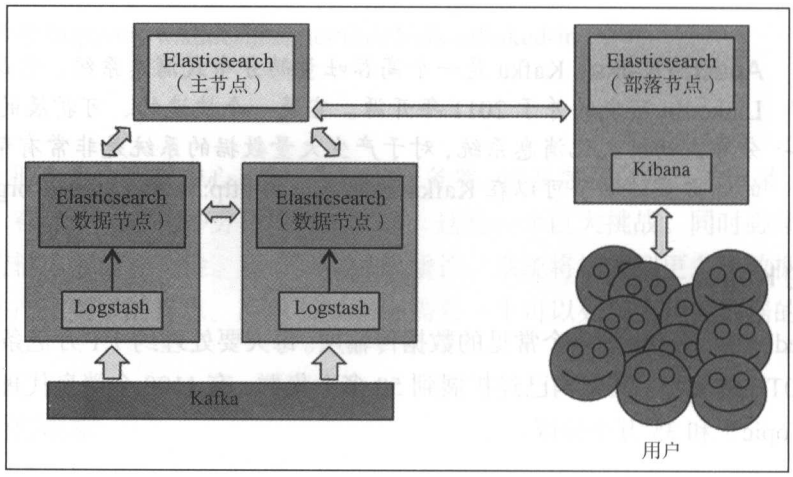
在 LinkedIn 中，Kafka 是一个常见的数据传输层。每天要处理约 1.1 万亿条消息、200TB 的输入和 700TB 的输出。其架构已经扩展到 50 多个集群，有 1100 个消息代理，包括约 3.2 万个主题（Topic）和 35 万个分区。

运维的挑战

LinkedIn 产生了大量数据，因此，可靠的传输、队列、存储和索引都是很重要的。还需要从不同来源采集数据，如 Java、Scala、Python、Node.js、Go 等。显然这些数据源的格式都是不同的，所以还需要进行转换。

LinkedIn 使用 Kafka 记录日志

LinkedIn 在每个数据中心为日志使用了专用集群。每个应用使用不同的 Kafka 主题，为所有服务、不同语言和框架提供了一个通用的日志传输服务。Logstash 从 Kafka 获取日志，最初用自己的 Kafka input 插件；后来采用 KCC（Kafka Console Consumer）的管道输入插件。



LinkedIn 的 ELK

使用 KCC Logstash 管道插件的样例配置如下：

```
pipe {
    type => "mobile"
    command => "/opt/bin/kafka-console-consumer/kafka-console-
consumer.sh \
     --formatter
com.linkedin.avro.KafkaMessageJsonWithHexFormatter \
     --property schema.registry.url=http://schema-
server.example.com:12250/schemaRegistry/schemas \
     --autocommit.interval.ms=60000 \
     --zookeeper zk.example.com:12913/kafka-metrics \
     --topic log_stash_event \
     --group logstash1"
    codec => "json"
}
```

SCA 使用 ELK 的案例

爱生雅（SCA）是一家全球领先的卫生用品与林业公司。集团研发并生产具有可持续性的个人护理用品、纸巾和林业产品。就在网站 https://www.elastic.co/blog/improving-user-

intelligence-with-the-elk-stack-at-sca 中可以看到以下介绍：

> “在 SCA，我们使用 Elasticsearch、Logstash 和 Kibana 记录用户在内部网站和外部网站中的搜索、对结果文档的点击行为及用户反馈。另外，我们还会收集一些定性指标，如在显示搜索结果页后会问用户一个类似“找到你要找的东西了吗”的问题，用户可以选择向上或向下的大拇指图标来表示赞同或否认，当然也可以直接写评论。”

SCA 如何使用 ELK

每个搜索事件都记录了所有搜索参数和结果信息，如查询字符串、分页、排序、维度、命中数、搜索响应时间、搜索日期和时间等。点击结果文档时也记录了大量的信息：文档在结果列表中的位置、从搜索到点击的时间，以及文档的元数据（如 URL、来源、格式、最后修改时间、作者等）。点击事件还会与产生该点击的搜索事件关联起来。反馈事件的情况也会做同样的处理。

Logstash 会实时监控写入日志文件的每个事件，为每个事件生成一个文档，并推送到 Elasticsearch，最后在 Kibana 中展示。

如何帮助分析

因为大量信息都已经索引到 ELK 技术栈中，所以通过简单的查询就能做各种分析，如“在过去的一周里，10 个最常见的搜索是什么”、“点击 X 文档的用户都在查找什么”，还有更复杂的，如“在每周三来源于 S 的被点击文档的最后修改时间的分布是怎样的”。

类似这样的分析可以帮助对搜索进行优化，以满足用户需求，并提供更大的价值。通过分析，可以调整相关性模型，可以增加新的维度或删除旧的维度，或者更改搜索页和搜索结果页的布局。

对 SCA 来说，这意味着可以持续地改进搜索。用户和系统管理员之间的直接反馈渠道可以形成一定的社区意识，尤其是用户，可以看到自己的建议正在被采纳。用户可以不断地找到自己想要的结果，从而节约时间并减少挫折感。

SCA 使用 ELK 做监控

ELK 不仅可以设置用于记录用户行为的信息，还可以用来监控服务器的健康状况。在监控场景中，Elasticsearch、Logstash 和 Kibana 可以当作时序数据库来使用。每隔几秒，服务器的 CPU、内存和磁盘使用数据（都是时序数据）将被索引。这可以用来查询历史数据并找出系统的趋势。当然，这类趋势也可能与用户行为相关。例如，CPU 使用率上升就可能与查询量的增加有关。

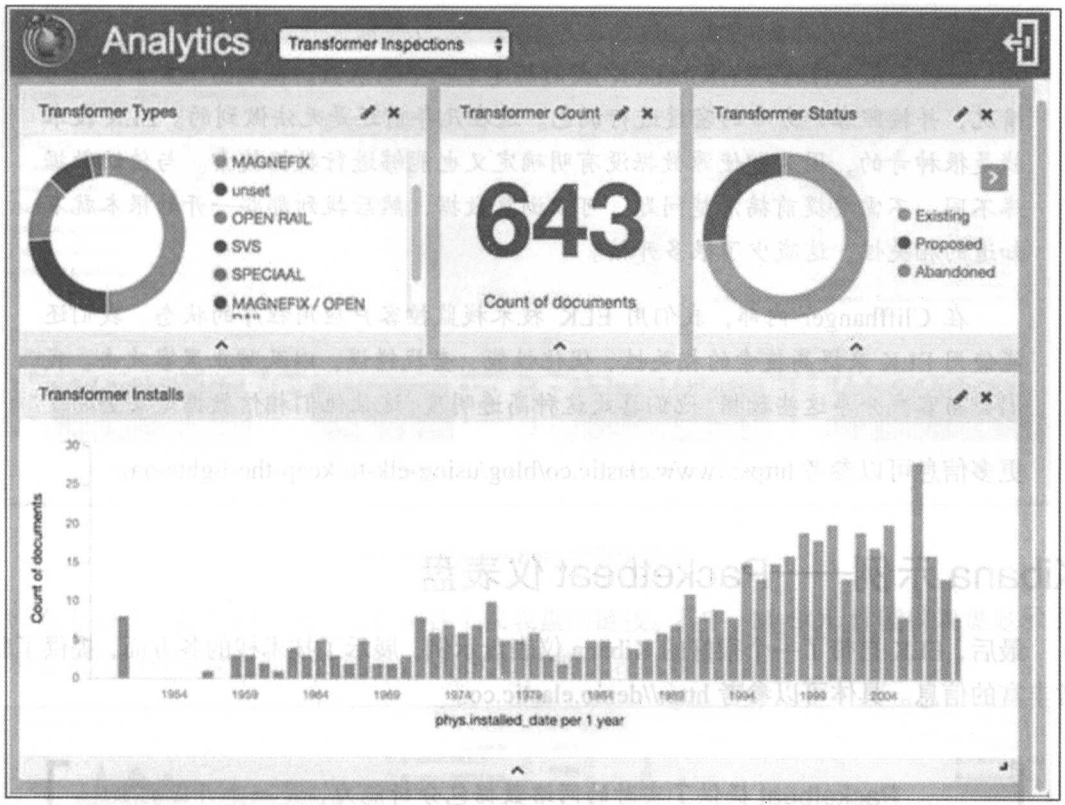
关于本案例，可以参考：

https://www.elastic.co/blog/improving-user-intelligence-with-the-elk-stack-at-sca。

Cliffhanger Solutions 使用 ELK 的案例

Cliffhanger Solutions 是公用事业和电信行业的应用服务提供商。它主要为客户和公用事业公司提供有计划的预防性维护，以帮助减少故障恢复时间。

> “在 Cliffhanger Solutions 中，我们使用 Elasticsearch 和 Logstash 对各种数据源的数据做实时索引。数据源包括维修车的 GPS 定位数据，或运行在平板电脑上的 App，以及智能仪表的读数和 GIS（地理信息系统）设备数据。”



Cliffhanger 的 Kibana 仪表盘

现在运维人员可以很快得到问题的答案，如“我是否可以安全地关闭这个开关，向 1500 个客户恢复电力供应”或“一场风暴正在从南方袭来，从风暴袭击的地方拿回我的吊斗车需要多久”。为了预防性维护，工程师可以搜索问题的答案，诸如“制造商 X 的变压器的 MTBF（平均故障间隔时间）高于平均值。找到所有这样的变压器，并以安装日期进行排序，然后将它们发送到工单系统进行检查或更换。”虽然听起来没什么大不了，但这确实是很难实现的。到目前为止，如果没有在咨询顾问身上进行重大投资，或者没有与一个能提供整体解决方案的一站式大型供应商进行绑定，这都是不可能实现的。

正如以下博客 https://www. elastic.co/blog/using-elk-to-keep-the-lights-on 中写到的:

“Cliffhanger Solutions 是一个小公司，但 Elasticsearch 的灵活性使得我们专注于为客户创造价值而不是为不同客户维护不同的系统。而且我们的客户也使用

ELK。例如，我们为一个在加勒比海岛上的小公司提供服务，IT 部门里只有 1 个人。通过使用 ATLAS（+ Kibana），我们创建了一个仪表盘，在地图上显示断供的情况，并按照客户分布的密度进行调色。这在几年前还是无法做到的。ELK 技术栈是很神奇的，因为即使源数据没有明确定义也能够进行数据搜索。与传统数据库不同，不需要提前搞清楚问题，可以浏览数据，然后找到那些一开始根本就不知道的相关性。这减少了很多开销。

在 Cliffhanger 内部，我们用 ELK 技术栈监控客户应用程序的状态。我们还能使用 ELK 来提高搜索的相关性、优化性能、查找错误，以及防止黑客攻击。我们会与客户分享这些数据。他们喜欢这种高透明度，这让他们相信数据是安全的。”

更多信息可以参考 https://www.elastic.co/blog/using-elk-to-keep-the-lights-on。

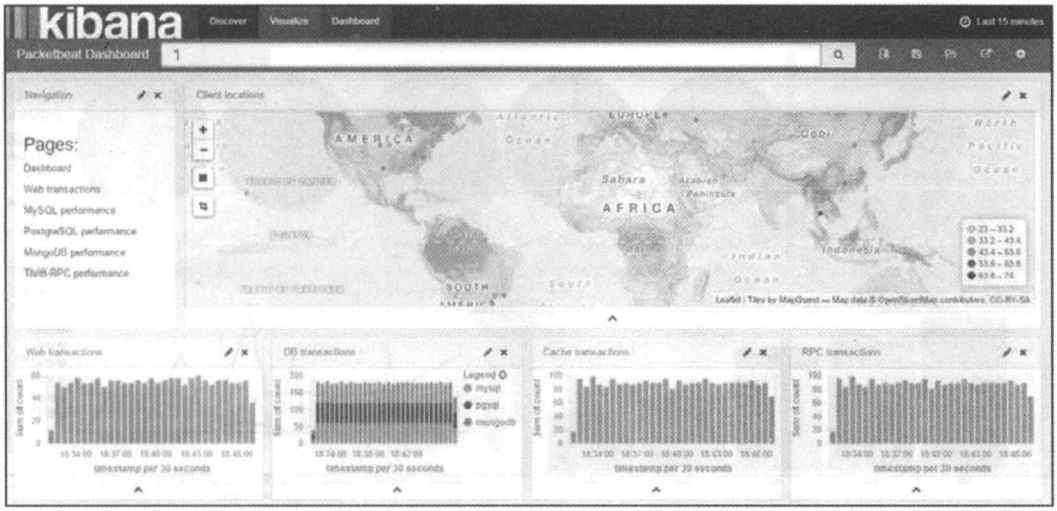
Kibana 示例——Packetbeat 仪表盘

最后，ELK 自带了一个很好的 Kibana 仪表盘示例，展示了技术栈的各方面，提供了非常丰富的信息。具体可以参考 http://demo.elastic.co。



Packetbeat 提供了实时的网络数据包分析能力，是一个开源的数据采集器，可以与 Kibana 和 Elasticsearch 集成，为 Web 网络、数据库及其他网络协议提供实时分析。

这个示例基于 Packetbeat 实现了多个仪表盘，如已经有 MySQL、MongoDB、Web 事务、Thrift-RPC 和 PostgreSQL 等不同的仪表盘。这个示例可以帮助我们了解很多基于 Kibana 平台的高级搜索和可视化。下图是示例仪表盘：



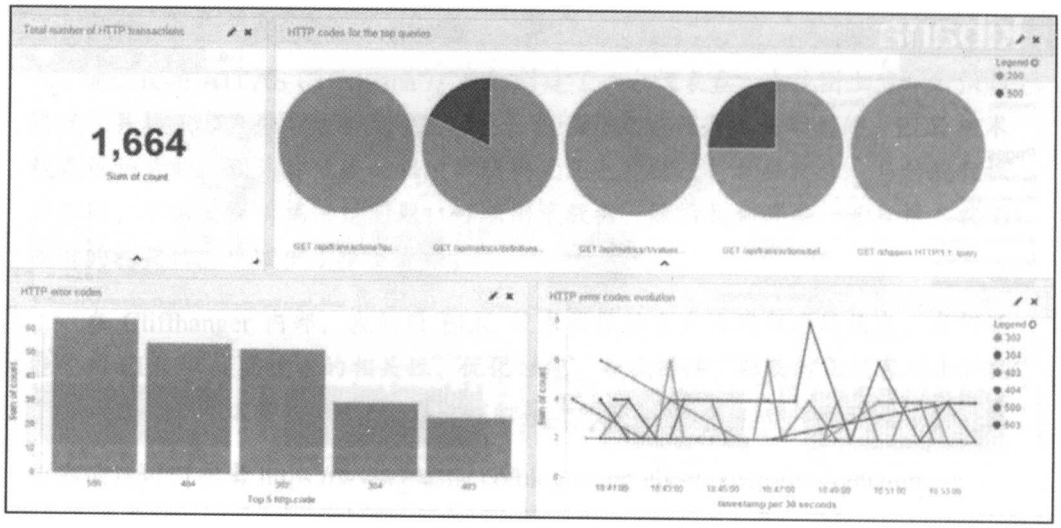
Kibana Packetbeat 示例仪表盘

在上面这个截图的左侧可以看到各个仪表盘的链接。其中，MySQL 性能仪表盘显示了各种使用过的查询及查询的性能等，MySQL 性能仪表盘如下：



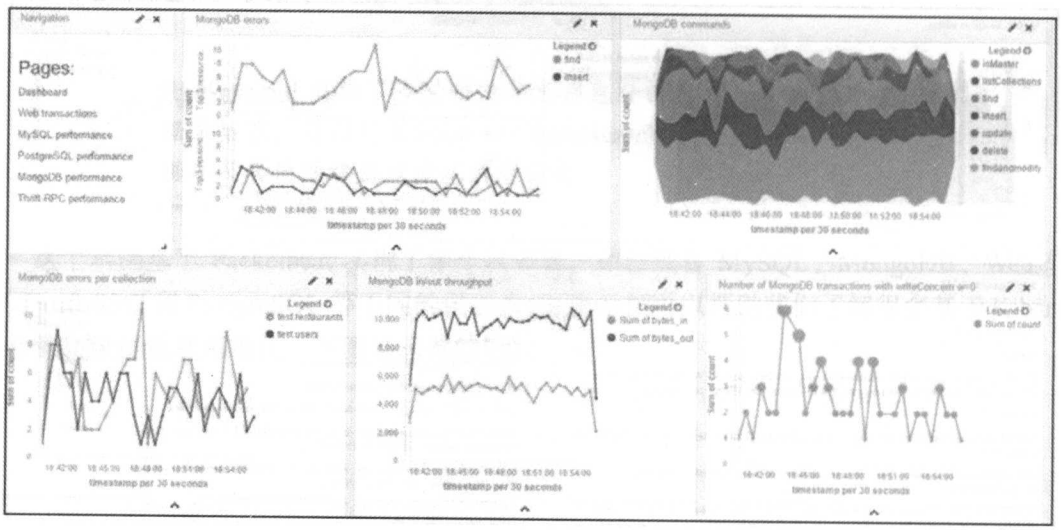
MySQL 性能仪表盘

Web 事务仪表盘显示了各种 Web 事务，包括使用不同的 HTTP 方法、总请求数、错误代码等，Web 事务仪表盘如下：



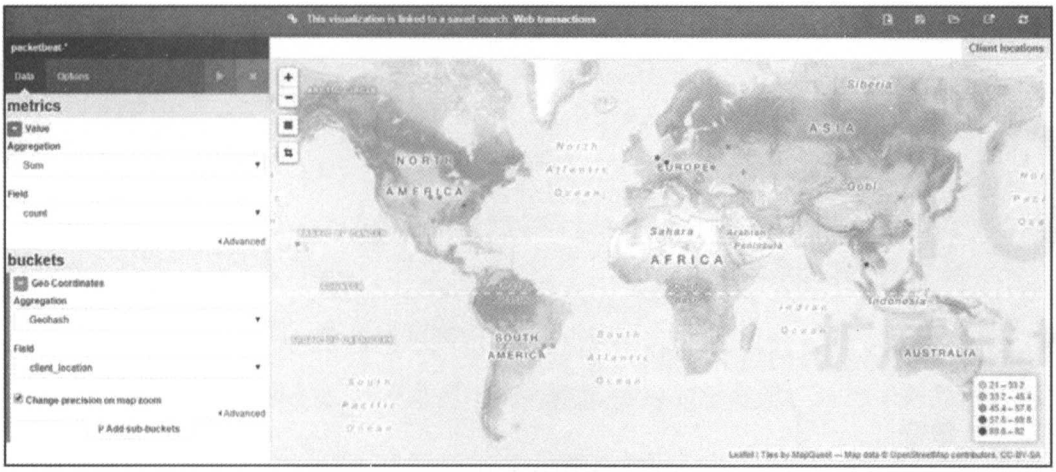
Web 事务仪表盘

MongoDB 性能仪表盘显示了 MongoDB 的吞吐量、错误数、每个 collection 的错误数、输入输出的吞吐量等，如下所示：



MongoDB 性能仪表盘

我们还可以探索构建这些仪表盘的多种可视化组件。例如，GeoIP 可视化的配置，可以像下图这样显示用户的地理位置：



Geo IP 可视化

总结

在本章中，我们研究了一些 ELK 技术栈在生产环境的策略，也学习了不少案例。随着社区用户的不断加入，以及大量从 ELK 技术栈中受益的案例，可以看到 ELK 技术栈的应用已经越来越广泛。

10

扩展 ELK

在之前的章节中，我们探讨了 ELK 技术栈的所有功能，以及如何利用 ELK 简化日志分析。现在，我们来探讨一些扩展 ELK 技术栈能力的插件和工具，使得 ELK 更精彩、更安全、更易于维护。最后再来看看 ELK 技术栈各组件的路线图。

本章将包括以下主题。

- Elasticsearch 插件和工具：
 - 用于索引管理的 Curator。
 - 用于安全的 Shield。
 - 用于监控的 Marvel。
- ELK 路线图。

Elasticsearch 插件和工具

Elasticsearch 是 ELK 技术栈中非常重要的组件，所以管理好集群并维护其安全性是非常重要的。有几款可用的插件和工具能够提供帮助。

用于索引管理的 Curator

Curator 是一个有助于管理 Elasticsearch 索引的重要工具。随着数据的增加，它变得非常重要，可以管理各种需要保留的数据和删除的数据。Curator 可以用来删除旧的索引并优化系统。

以下是一些 Curator 可以在 Elasticsearch 索引上执行的高级任务：

- 删除索引和快照。
- 关闭索引。
- 打开已关闭的索引。
- 显示索引和快照。
- 从别名中添加或删除索引。
- 索引优化。
- 更改索引分片的副本数量。

Curator 命令

Curator 可以简单地配置成系统的 Cron 任务，来执行定期的索引清理。让我们来看看 Curator 的命令行语法：

```
curator [FLAGS] COMMAND [FLAGS] SUBCOMMAND [FLAGS]
```

使用"help"命令可以查看所有可用的选项：

```
curator --help
```

让我们来看以下如何使用 Curator 例子：

- 删除某个时间之前的索引：

```
curator --host 10.0.0.X delete indices --older-than 180 --time-
unit days \ --timestring '%Y.%m.%d'
```

- 显示匹配时间字符串的所有索引：

```
curator --host 10.0.0.x show indices --timestring '%Y.%m.%d'
```

- 将 30 天之前的索引添加别名为 last_month：

```
curator alias --alias-older-than 30 --alias last_month
```

- 从别名 last_month 中删除 60 天以前的索引：

```
curator alias --unalias-older-than 60 --alias last_month
```

安装 Curator

安装 Curator 非常简单快捷；可以通过 python pip 工具来完成：

```
pip install elasticsearch-curator
```

Curator 项目托管在 https://github.com/elastic/curator 中，详细信息可以在其官方文档中找到：https://www. elastic.co/guide/en/elasticsearch/client/curator/current/index.html。

用于安全的 Shield

Shield 是 Elastic 提供的一个用于集群安全的 Elasticsearch 插件。Shield 使用安全认证或基于角色的授权流程保护数据。

下面是 Shield 提供的一些高级功能：

- 通过启用密码保护、基于角色的访问控制和 IP 过滤技术，增加了对集群的授权控制。
- 支持 SSL/TLS 加密，并且支持消息认证的功能。
- 增加了审计功能，维护了数据变更的设计跟踪情况。



关于 Shield 的更多细节，可以在官方文档中找到：
https://www.elastic.co/guide/en/shield/ current/index.html。

Shield 提供了 30 天的试用许可，如果 30 天之后继续使用，则需要订阅购买。

也有一些可替代 Shield 的开源产品，如 Search Guard（https://github.com/floragunncom/search-guard）。

安装 Shield

如果要安装 Shield，可以在 Elasticsearch 的安装目录下执行如下操作。

1. 安装 license 插件：

```
bin/plugin -i elasticsearch/license/latest
-> Installing elasticsearch/license/latest...
Trying http://download.elasticsearch.org/elasticsearch/license/
license-latest.zip...
Downloading ..................................................
DONE
Installed elasticsearch/license/latest into /usr/share/
elasticsearch/plugins/license
```

2. 安装 shield 插件：

```
bin/plugin -i elasticsearch/shield/latest
-> Installing elasticsearch/shield/latest...
Trying http://download.elasticsearch.org/elasticsearch/shield/
shield-latest.zip...
Downloading
DONE
Installed elasticsearch/shield/latest into /usr/share/
elasticsearch/plugins/shield
```

3. 插件安装完成后，启动 ElasticSearch 实例，检查 shield 相关日志：

```
[2015-10-17 07:46:27,508][INFO ][transport                ]
[Witchfire] Using [org.elasticsearch.shield.transport.
ShieldServerTransportService] as transport service, overridden by
[shield]
[2015-10-17 07:46:27,510][INFO ][transport                ]
[Witchfire] Using [org.elasticsearch.shield.transport.netty.
ShieldNettyTransport] as transport, overridden by [shield]
[2015-10-17 07:46:27,511][INFO ][http                     ]
[Witchfire] Using [org.elasticsearch.shield.transport.netty.
ShieldNettyHttpServerTransport] as http transport, overridden by
[shield]
```

添加了 shield 插件后，再通过 `http://localhost:9200` 访问 Elasticsearch，没有通过认证的请求就会被限制访问。

增加用户和角色

访问 Elasticsearch 之前，需要在 shield 中添加用户和角色。执行以下命令可以添加一个用户和角色，并且可以为每个用户设置密码：

```
bin/shield/esusers useradd es_admin -r admin
```

```
packtpub@saurzcode:/usr/share/elasticsearch/bin/shield$ sudo ./esusers useradd es_admin -r admin
Enter new password:
Retype new password:
```

在 shield 中添加角色

添加完成后，可以通过 `list` 命令来验证用户、删除用户、更改密码等。

```
packtpub@saurzcode:/usr/share/elasticsearch/bin/shield$ sudo ./esusers list
es_admin        : admin
packtpub@saurzcode:/usr/share/elasticsearch/bin/shield$ sudo ./esusers userdel es_admin
packtpub@saurzcode:/usr/share/elasticsearch/bin/shield$ sudo ./esusers list
No users found
```

在 shield 中列出和删除角色

注意：我们安装的是 Shield 插件的 30 天试用 license 插件，过期之后功能会被降级，试用受限，只有购买正式许可后才能激活所有功能。

使用 Kibana4 访问受 shield 保护的 Elasticsearch

如果要使用 Kibana4 来访问配置了 shield 插件的 Elasticsearch 实例，需要在 shield 中添加 `kibana4-server` 角色，并在 Kibana 安装目录下的 `config/kibana.yml` 文件中配置相关内容。

以下是 Kibana 服务角色：

```
esusers useradd kibana4-server -r kibana4_server -p password
```

以下是 Kibana 配置：

```
kibana_elasticsearch_username: kibana4-server
kibana_elasticsearch_password: password
```

用于监控的 Marvel

Marvel 是用于监控 ElasticSearch 集群的产品。它提供了一个单一的界面来查看群集的聚合分析。在这里可以查看集群的一些关键指标，如健康状况、节点状态和索引等。Marvel 对集群相关的问题做根源分析非常有帮助，确保可以提前发现问题并解决。也可以用它来分析历史数据或实时性数据。



Marvel 2.0 支持 Elasticsearch 2.0 版本，已经完全重构成了 Kibana 插件。个人用户可以免费使用，但多集群的支持需要商业许可。关于 Marvel 2.0 的更多信息可以在这里找到 https://www.elastic.co/guide/en/marvel/current/index.html。

安装 Marvel

正如 shield 的安装过程，Marvel 的安装也是一步即可。在 Elasticsearch 的安装目录下执行以下命令：

```
bin/plugin -i elasticsearch/marvel/latest
```

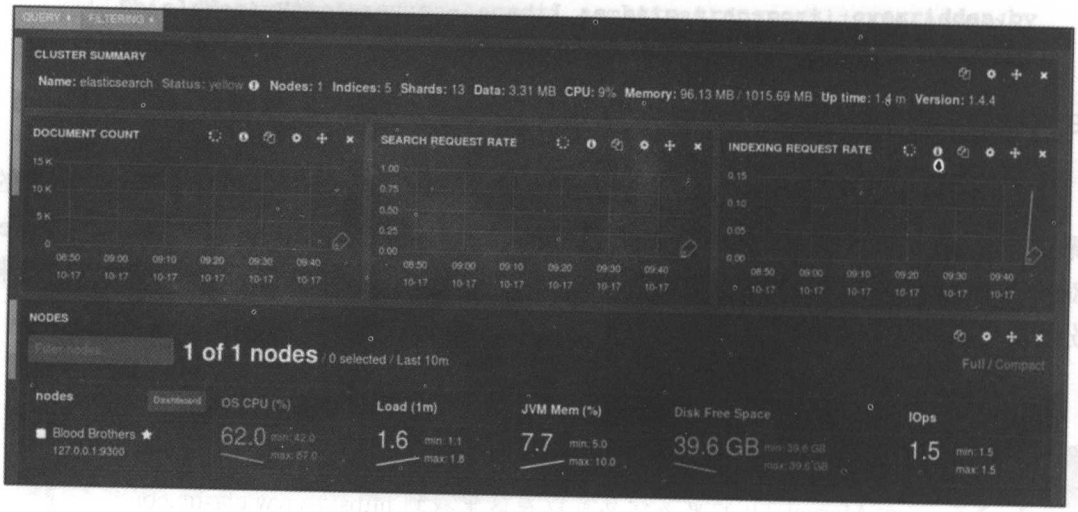
```
packtpub@saurzcode:/usr/share/elasticsearch/bin$ sudo ./plugin -i elasticsearch/marvel/latest
[sudo] password for packtpub:
-> Installing elasticsearch/marvel/latest...
Trying http://download.elasticsearch.org/elasticsearch/marvel/marvel-latest.zip...
Downloading ..........
.....
.....
.....
.....
.....
.....
.....
.....DONE
Installed elasticsearch/marvel/latest into /usr/share/elasticsearch/plugins/marvel
```

安装 Marvel

下面是 Marvel 所提供的一些功能。

Marvel 仪表盘

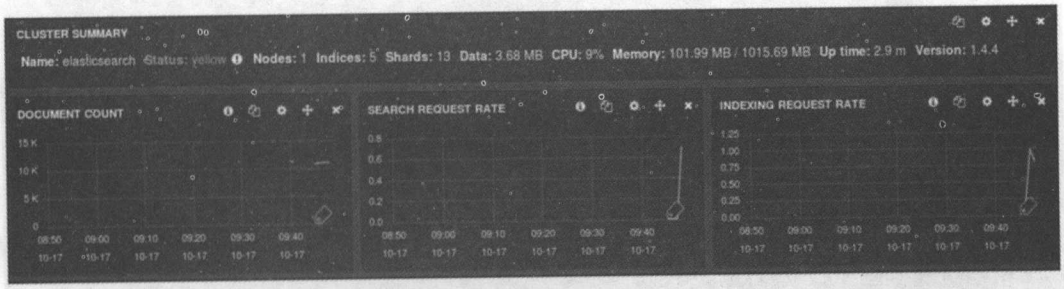
Marvel 仪表盘看起来与 Kibana 仪表盘非常像，它提供了 Elasticsearch 集群的各种指标，以及节点和索引的信息。其中以黄色显示的值需要引起注意，必须处理好。



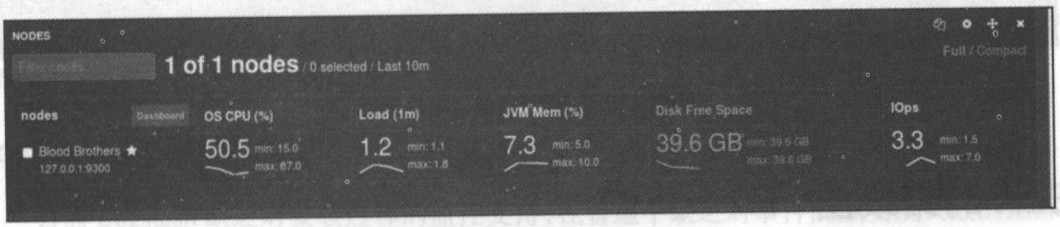
Marvel 仪表盘

它提供了文档数量、搜索请求率、索引请求率，以及节点和索引的各种统计指标，如

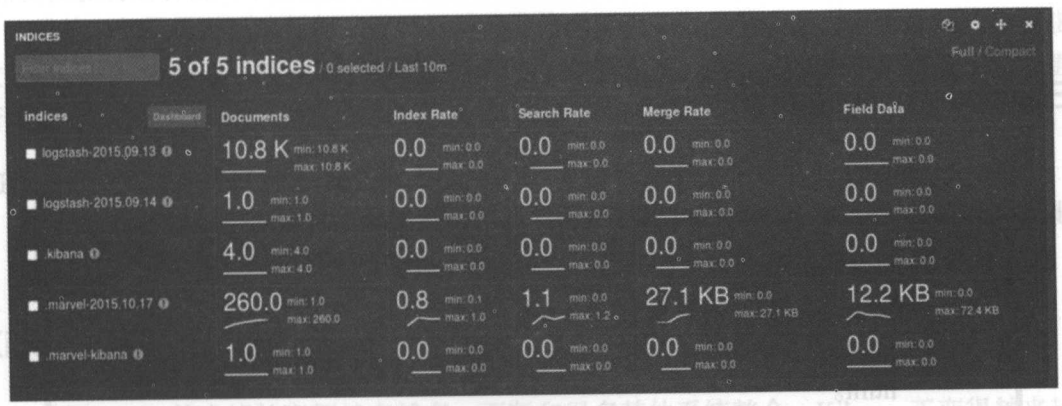
系统 CPU、负载、JVM 内存、磁盘可用空间和操作的 IOPS 等，如下图所示：



Marvel 仪表盘统计数据

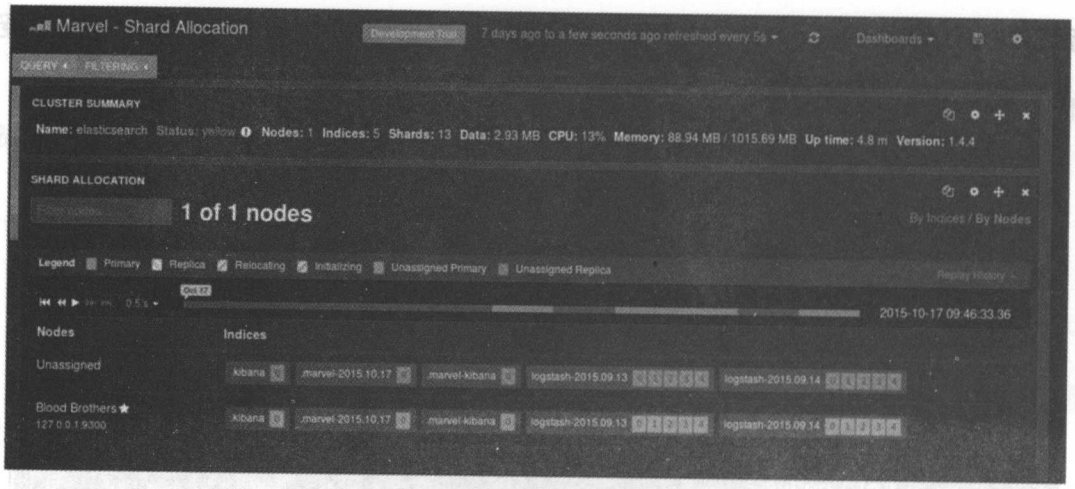


Marvel 节点指标



Marvel 索引指标

还有一个很好的仪表盘，可用于展示群集分片的分布情况，不同的索引各有不同种类的分片和副本，并用不同颜色显示了所有主节点和副节点的节点状态。



Marvel 分片的分布情况的仪表盘

ELK 的路线图

所有 ELK 技术栈的生态系统中的工具都处在积极的发展阶段中，新特性会定期发布。ELK 的迅速发展可以适应现代企业应用的需要。

Elasticsearch 路线图

Elasticsearch 已经在各种企业应用场景下广泛使用，其生态系统和插件的发展非常迅速。涉及 Elasticsearch，有各式各样的丰富的插件可用，适用越来越多的使用场景。



所有插件和整合文档都在这里：
https://www.elastic.co/guide/en/elasticsearch/plugins/current/index.html。

Logstash 路线图

在 ELK 技术栈中，Logstash 可能是最活跃的开发工具。开发团队和社区正在努力为 Logstash 增加更多的弹性、健壮性、可维护性，使其更适应于企业级应用。

Logstash 1.5.x 中插件的管理和开发已经发生了重大变化。下面是 Logstash 未来版本中将要添加的功能。

事件的持久化能力

目前需要使用消息中间件（如 Redis 等）对事件队列进行限流或防止管道中的事件丢失。在接下来的 Logstash 版本中，将添加事件队列持久到磁盘的功能，以避免在插件崩溃或重启的情况下丢失数据。

端到端消息确认

目前 Logstash 无法做到端到端的消息确认。未来的 Logstash 版本计划包含对失败事件的通知功能等，这样事件可以重新发送或处理。

Logstash 监控及管理 API

目前 Logstash 缺乏对安装过程的监控支持，在管道中缺乏对事件的成功或失败的跟踪。未来版本的监控 API 计划将提供这些功能。

另外，如果需要更改任何配置，目前需要修改 Logstash 配置文件并且重启系统才能生效。Logstash 的管理 API 规划会解除这个限制，这样就可以在不中断管道的情况下动态更新配置。



更多添加到 LogStash 的功能可以跟踪：https://github.com/elastic/logstash/labels/roadmap。

Kibana 路线图

采用 ELK 技术栈的案例越来越多，不断和很多其他系统整合，Kibana 正变得越来越有用。随着对各种数据进行分析的使用场景的增加，新的图表类型和聚合功能也在陆续增加。以下是最近要添加到 4.1 版本的一些功能：

- 有能力从**折线图**中衍生出泡沫图。
- **设置**页面的字段格式选项。
- Kibana 对象（仪表盘、图表和搜索）现在也可以导入/导出。



可以在下面的网站跟踪 Kibana 更多的改进点：
https://www.elastic.co/guide/en/kibana/ current/whats-new.html。

总结

在本章中，我们了解了一些在使用 ELK 技术栈时可以简化工作的工具和实用程序。此外，还探讨了 ELK 技术栈及其生态系统如何扩展支持多个系统和数据源，以不断适应现代企业的需要。

Broadview
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

Learning ELK Stack 中文版



ELK技术栈，包括Elasticsearch、Logstash和Kibana，是一组强大的开源工具的组合。Elasticsearch用于深度搜索和数据分析。Logstash用于日志集中、增强和解析。Kibana则实现了强大而漂亮的数据可视化。简单来说，ELK技术栈使得搜索和分析数据变得前所未有的简单。

本书将介绍ELK技术栈，首先演示了如何安装和配置整个技术栈，然后描述了如何使用技术栈构建首个数据管道。接着会探索Logstash的关键特性及其在技术栈中的作用，包括如何创建Logstash的插件，这样就可以使用自定义的插件来完成工作。Elasticsearch和Kibana在ELK技术栈中的重要性也会有所覆盖，包括不同类型的高级数据分析和一组不同的图形、表格和地图等。最后，在充分理解每个组件的功能的基础上，在本书的结束部分将可以使用ELK技术栈构建完整的数据管道。

本书面向的对象

如果你是开发或者DevOps工程师，对于使用ELK这样的开源技术栈基于不同格式和类型的数据源来构建一套迷人的洞察商业指标的系统有兴趣，那么这本书就非常适合你。如果有Unix或者任何一种编程语言的基本知识，则对于阅读本书非常有帮助。

你将从本书学习到

◎安装、配置并运行Elasticsearch、Logstash和Kibana

◎理解日志分析的需求，以及当前在日志分析中的挑战

◎使用ELK构建自己的数据管道

◎熟悉Logstash的关键特性及其提供的不同的输入、过滤和输出插件

◎构建自定义Logstash插件

◎使用Kibana的图表、直方图和快速搜索等特性创建有价值的分析

◎理解Elasticsearch在ELK技术栈中的作用

[PACKT]
PUBLISHING

上架建议：数据处理

ISBN 978-7-121-28884-5

9 787121 288845

定价：65.00元

PHEI
责任编辑：张春雨
封面设计：李　玲